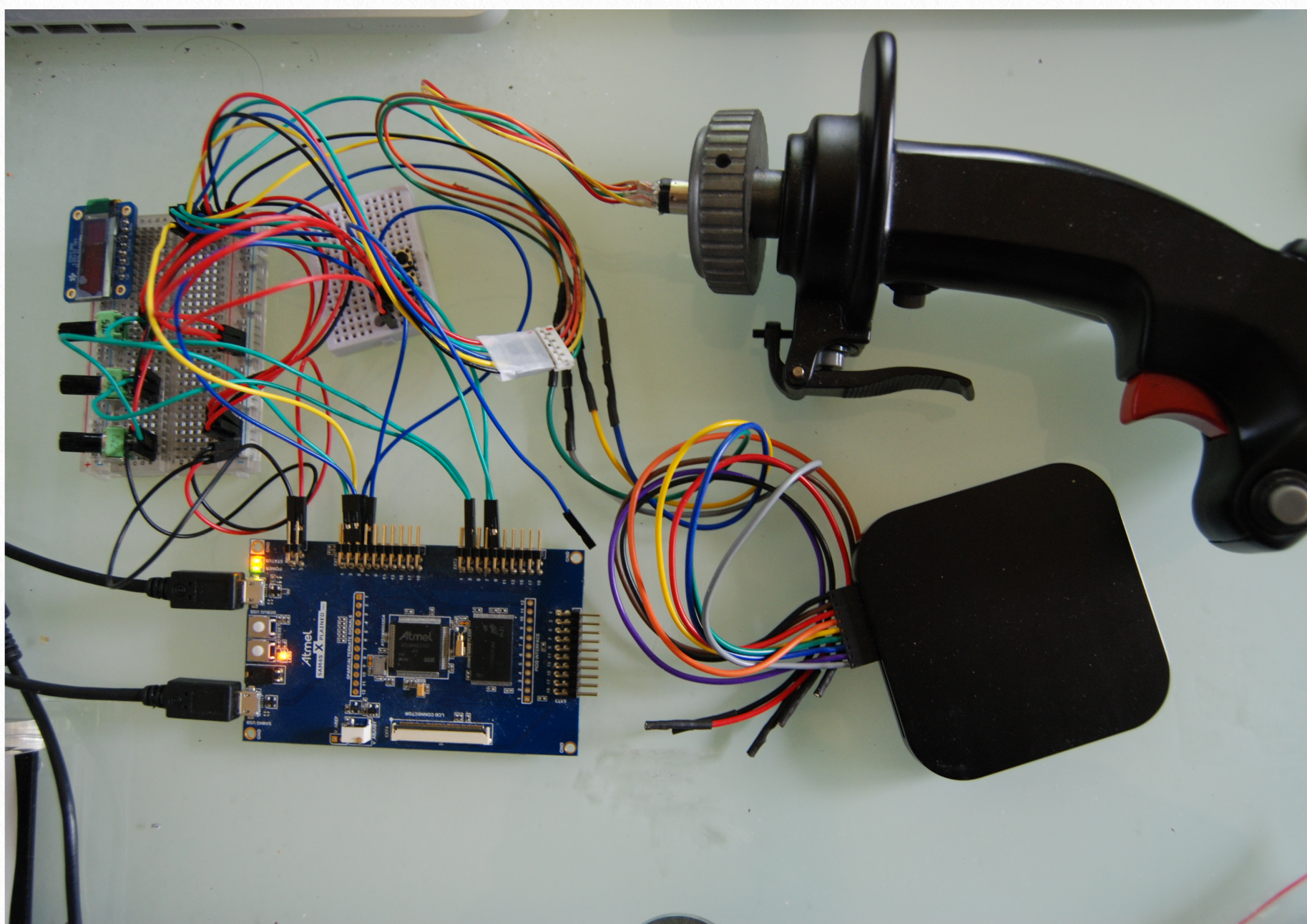


Jonah Tsai

Hempstick Rudder Demo

v.1.0.1



1

Overview

The purpose of this demonstration project & document is to show you how the basics of how to build a custom Hempstick rudder controller using the Atmel SAM4S XPLAINED Pro board to read 3x potentiometers (or any voltage source with voltage range of 0 to 3.3V, like Hall Sensors).

We will assume that you have already installed Atmel Studio v.6.2 or later by following the instructions in the Hempstick User Guide. We will also assume that you have already downloaded the libHemp and Hempstick and opened them in a directory on disk (both must be under the same directory to avoid having to change the project structure inside Atmel Studio).

We will demonstrate how to configure the Hempstick configuration files, wire up the potentiometers, compile and burn the firmware to the SAM4S XPLAINED Pro board, and then show you that the rudder shows up in ThrustMaster TARGET software.

However, due to the fear of DMCA and getting sued, I will omit the crucial part of how to configure it in order for it to be accepted by the TARGET software. You can either figure out this part yourself, or you can use vJoy/UJT instead of TARGET.

Why?

Why would you use Hempstick for your rudder?

1. If you have an old game port rudder.

2. Your old rudder has 10 bit (or goodness forbid, 8 bit) resolution. Hempstick is 12 bit, software oversampled to 14bit.
3. Your rudder does not work with TARGET.
4. You intend to use Hempstick to control more than just the rudder. For instance, you want to convert your old Cougar to use Hempstick, while also use the same Hempstick board to control your rudder.
5. Just to get some experiences of using Hempstick to build other custom controllers in the future. In this case, a rudder is the simplest starting point to cut your teeth.

What You Need

- Atmel Studio v.6.2 or newer installed.
- MSysGit & SmartGit (optional)
- libHempstick & Hempstick source code
- An Atmel SAM4S XPLAINED Pro board
- A rudder with potentiometers or hall sensors that outputs 0 - 3.3v (does not matter whether it's a USB or GamePort rudder, we will rip out all the electronics)
- Some wires and connectors to connect the potentiometers to the SAM4S XPLAINED Pro board
- Tools to do the wiring (that depends on the methods you decide to do the wiring)

2

Preparations

In this chapter, we will make sure that you have all the software tools installed and configured correctly.

Hempstick & libHemp Source Code

If you have not downloaded [Hempstick](#) & [libHemp](#) yet, download the zip files now. And unzip them into the, say `c:\workspace\HempstickDemo\`. From here on in this document, I will use `c:\workspace\HempstickDemo\` as the workspace directory, and all relative directory and file references will be assumed to be under this directory unless we spell out from `c:\`. You must make sure they have the path of `c:\workspace\HempstickDemo\Hempstick\` and `c:\workspace\HempstickDemo\libHemp\`.

Using the zip file downloads are the easiest way to get you started. But, if you modify anything, and there is a new release in the Hempstick GitHub repository, you will be forced to manually “merge” them. That is quite a pain, and sometimes a mission impossible.

Therefore, I highly recommend that you use Git revision control system, like I do. This way, if there is any update on Hempstick GitHub repository, you can simply issue a pull command. If there is any conflict, it will tell you about it, then you can try to resolve the conflicts.

If you want to use Git, I highly recommend that you use [MsysGit](#) & [SmartGit](#). SmartGit is a very friendly GUI built on top of Git. Git itself is written with command line in mind. But, the

command line arguments are so complicated that even a Unix command line guy like me try to avoid that when I can. Hence the SmartGit.

For the step-by-step instructions of installing MsysGit & SmartGit to get Hempstick source code, please see [Appendix A](#).

Opening the Hempstick Project & Testing the EDBG Driver

We have to make sure the Atmel Studio actually installed your USB drivers for EDBG correctly. The way that the Atmel EDBG works is that there is an additional EDBG chip in the back of the SAM4S XPLAINED Pro board. This chip connects to the SAM4S chip's debug port to control the SAM4S during debugging/programming.

There are two micro USB connectors on the board. One is labeled Debug USB, the other labeled SAM4S USB. The SAM4S USB connector is connected to the USB port on the SAM4S chip, and is what you would use to connect to your computer for it to show up as a Hempstick controller. The Debug USB connector is actually connected to the EDBG chip, not the SAM4S chip.

When you connect the Debug USB to your host computer, Windows will automatically loads up the Atmel EDBG driver, and your Atmel Studio will then know that there is an EDBG USB device in the system. All debugging/programming commands will be send from Atmel Studio via USB to the EDBG chip on the board. The EDBG chip then issues debug/programming commands to the SAM4S chip.

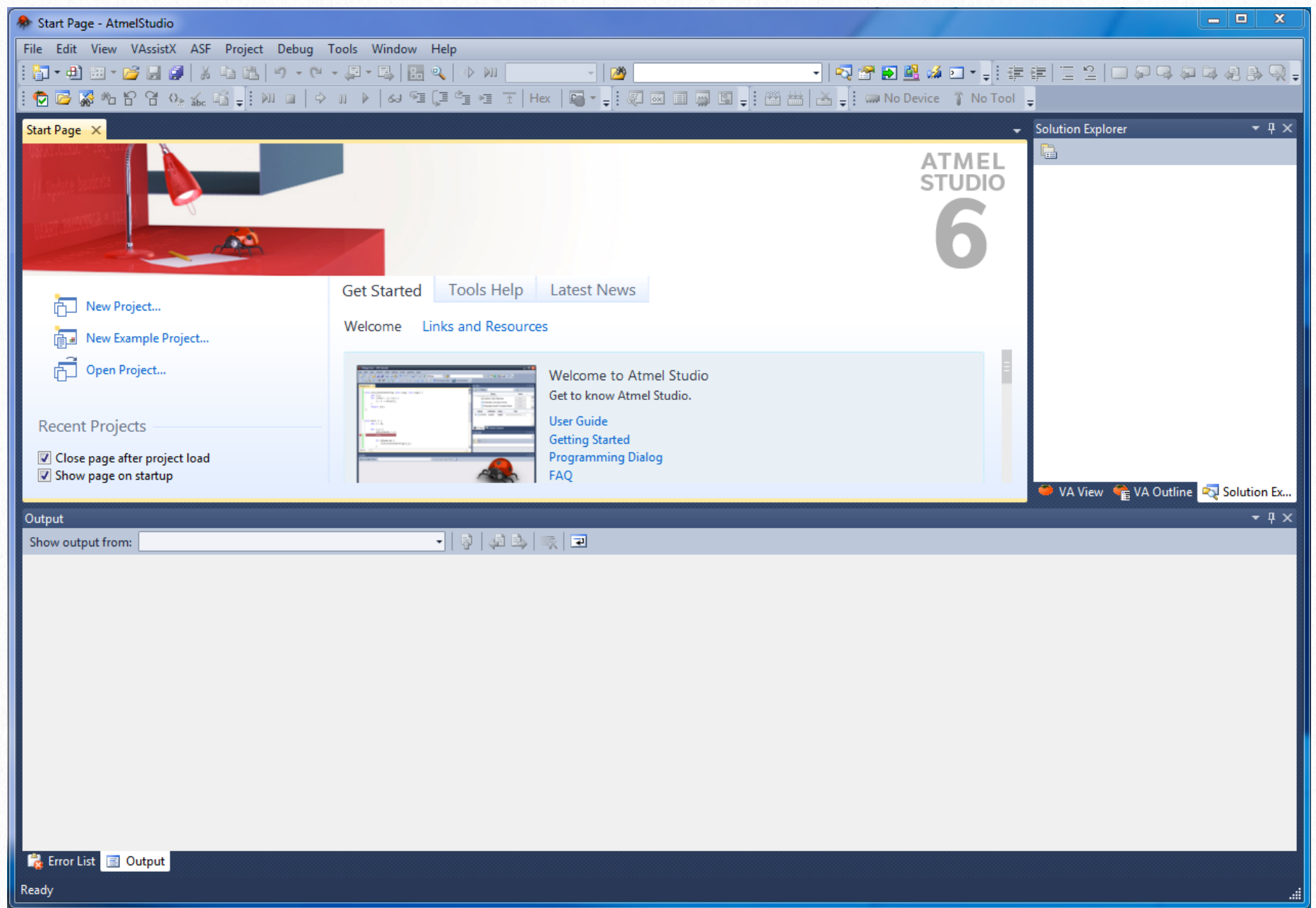
The EDBG thing is wonderful! We used to pay thousands of dollars for hardware debuggers per seat! Now it comes free with a USD \$39 board! Sure, it's not as powerful as the now hundreds, if not thousands, of dollars standalone debugger, but it works fine for developing Hempstick!

But, this EDBG thing is so new that it's a bit buggy, particularly the USB driver for it. It somethings hangs if you do the wrong thing in the wrong order, like disconnecting the De-

bug USB cable while you are in a debuggin session. Don't do that! The only way to fix it is to reboot the Windows OS. Adding insult to injury, your Windows OS will not shutdown correctly when that happens. It will get stuck in the "Shutting down....." screen forever, necessitating hitting the big red button!

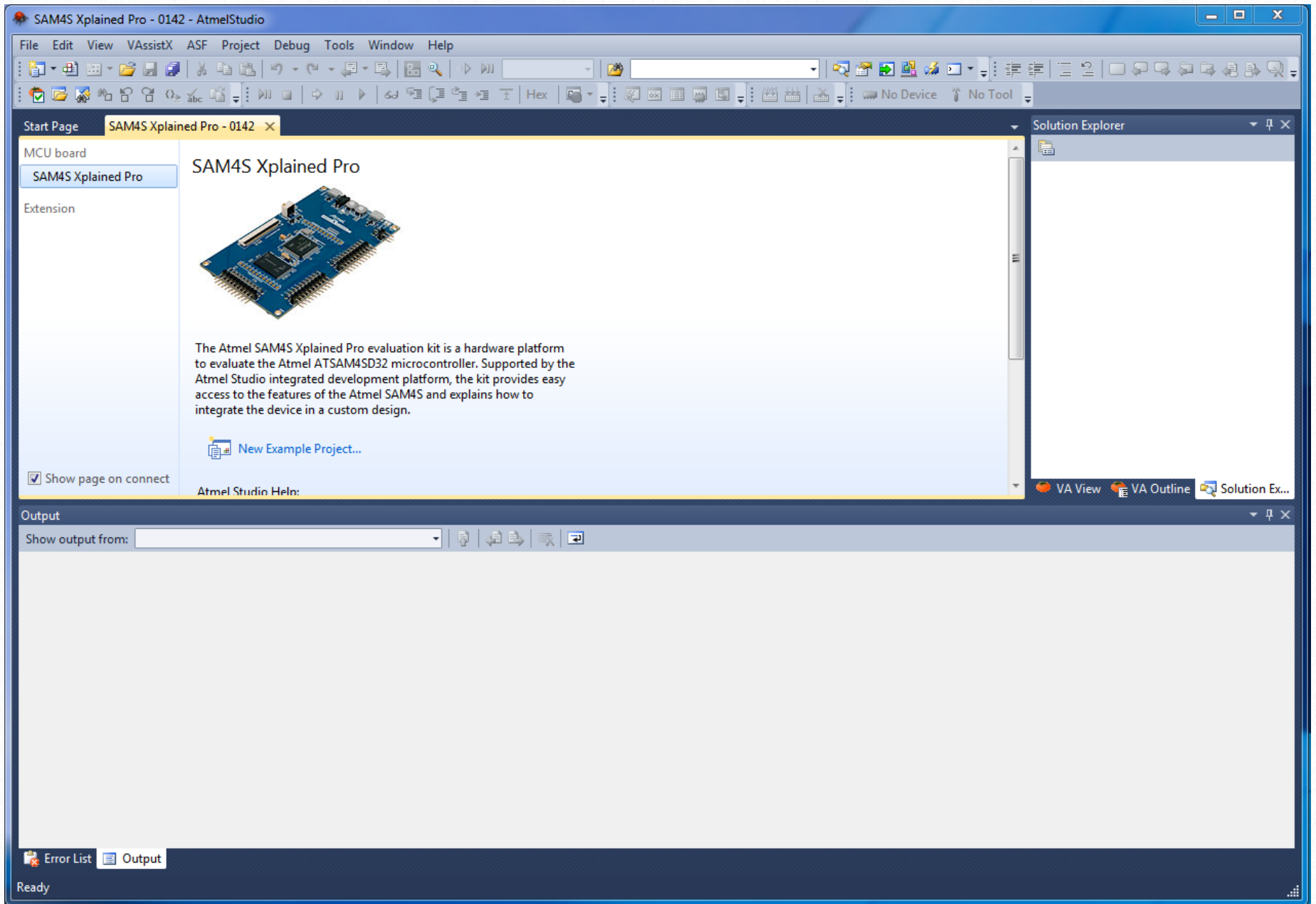
Enough of the gripe about the bugs in Atmel EDBG. Now, launch the Atmel Studio.

You should see the Start page.



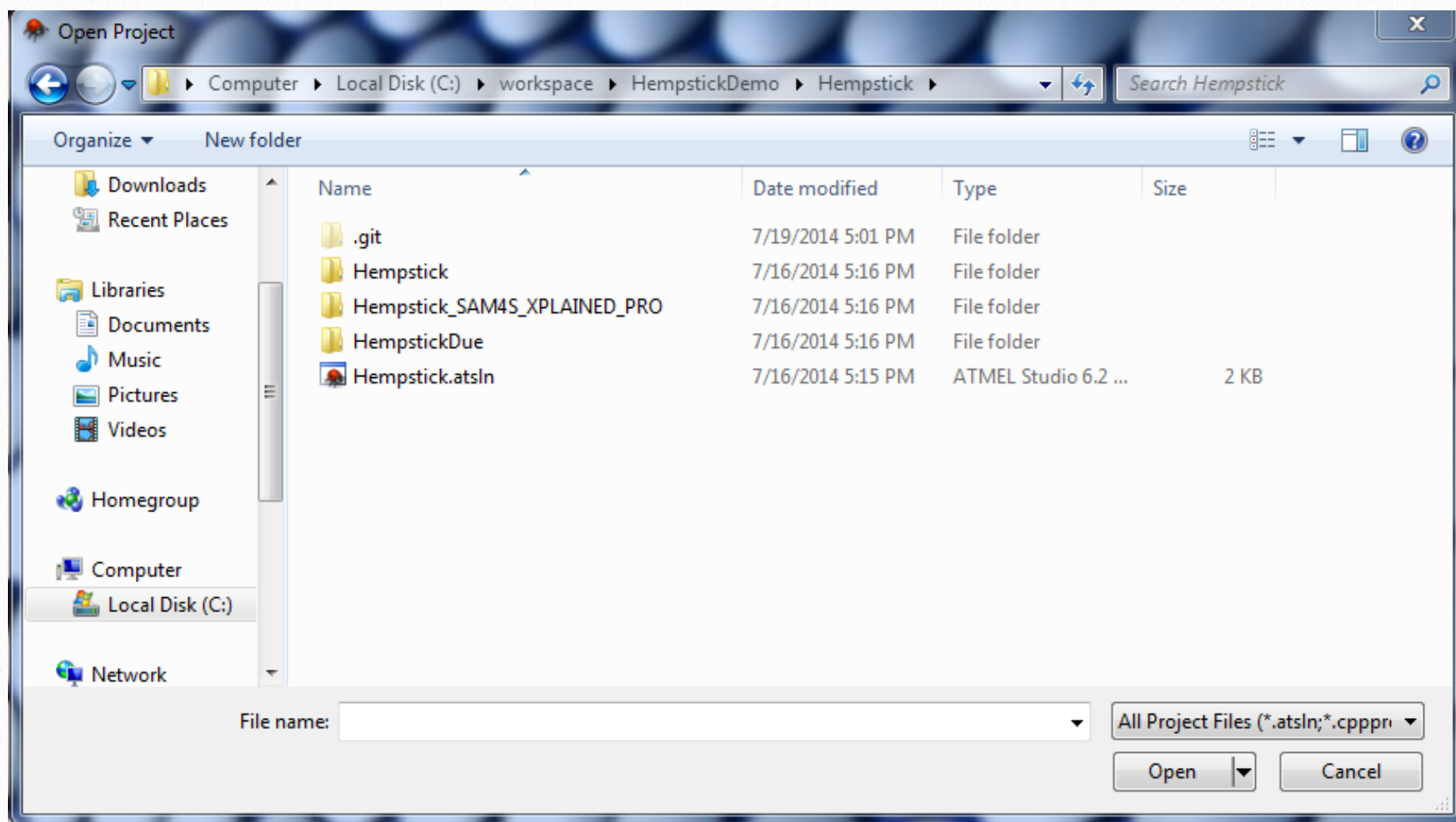
Now, plug in a micro USB cable into the Debug USB connector on the SAM4S XPLAINED Pro board, and then plug the other end into the host development computer. You should hear a ding, and Windows starts loading the USB driver for the EDBG device.

Once that is done correctly, your Atmel Studio should then show you the SAM4S XPLAINED Pro page. This means the EDBG connection to the Windows works somehow, at least the EDBG chip is reporting to Atmel Studio correctly about the board.

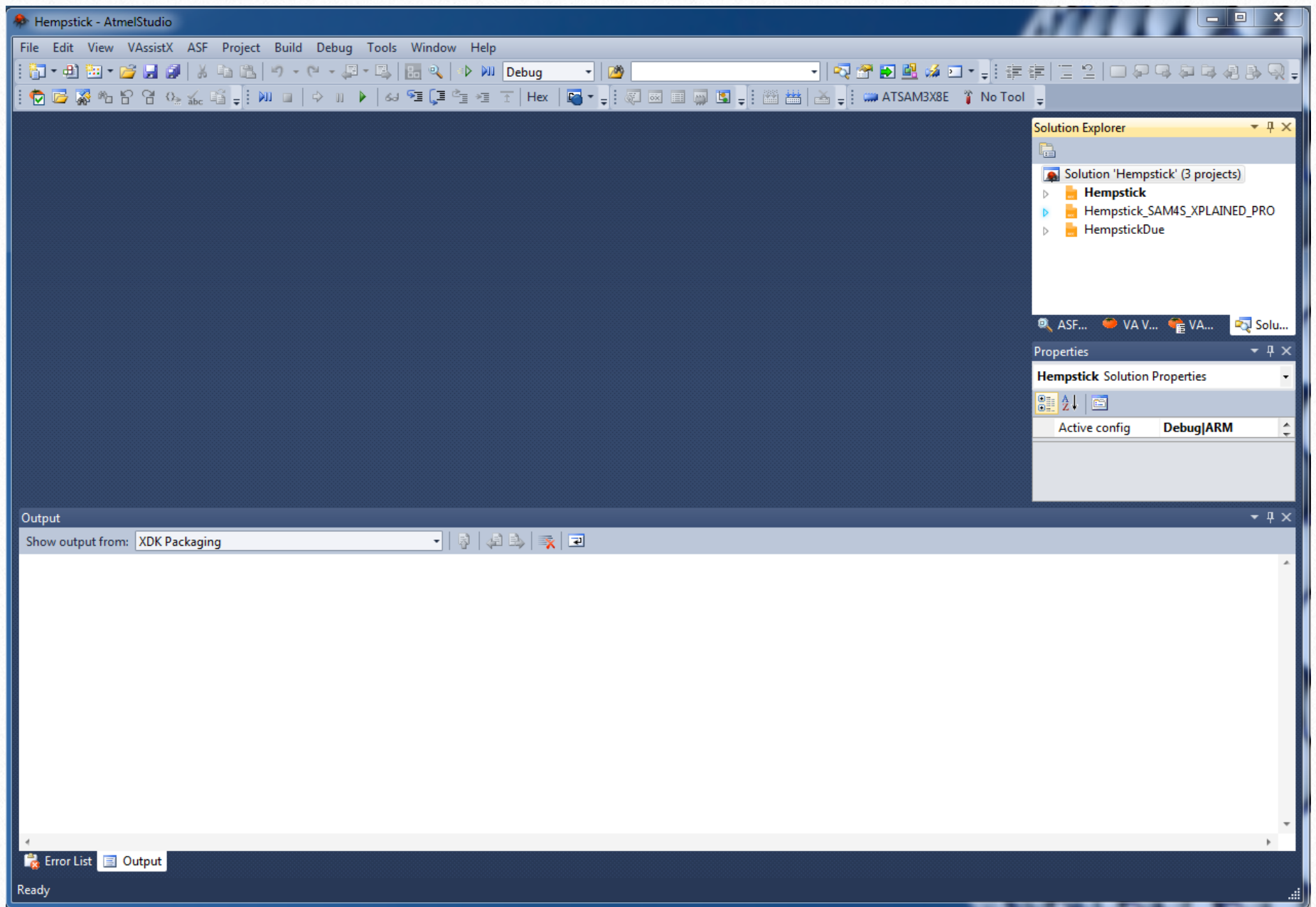


Close the Start page and the SAM4S page. They are irrelevant to us for now.

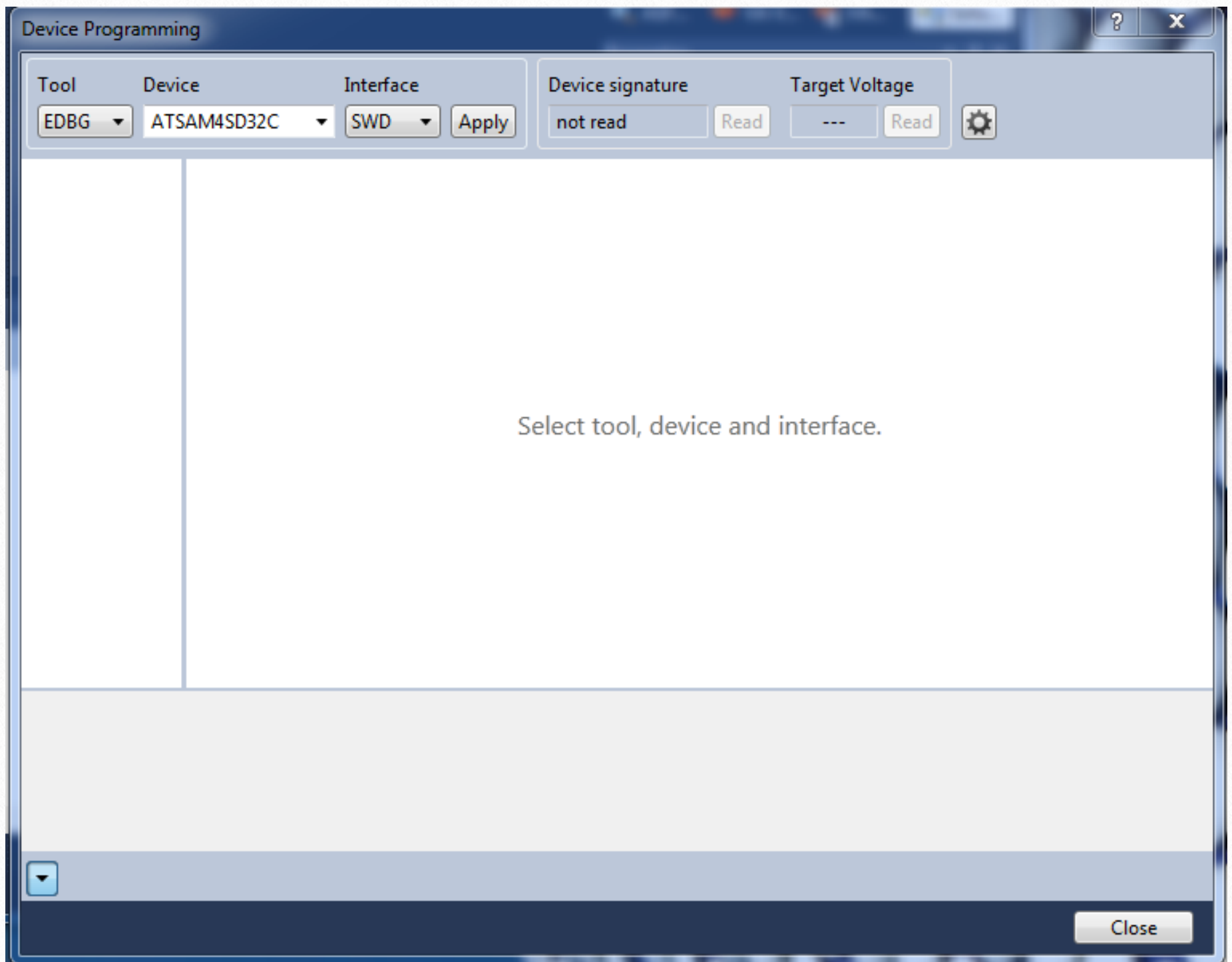
Go to the menu and select [File] -> [Open] -> [Project/Solution...]. There should be a popup window for file selection. Navigate to the c:\workspace\Hempstick\ and select the Hempstick.atsIn file, and click [Open].



You should see the following screen after loading the “solution.”



Now, on the right Solution Explorer panel, select the Hempstick_SAM4S_XPLAIN_Pro project. And then, go to the menu, select [Tools] -> [Device Programming].



You should be able to see under the upper left corner [Tool] drop down box, there are at least two choices. One is EDBG.... and the other is Simulator. The EDBG... is obviously the EDBG chip on the SAM4S XPLAINED Pro board, and the other one is the software chip simulator. Yes, you can save some time for some of your coding by using the software simulator, but there are things that it cannot simulate, USB is one of them. Well, at least it cannot simulate USB wire signals to the host computer to simulate a real USB device.

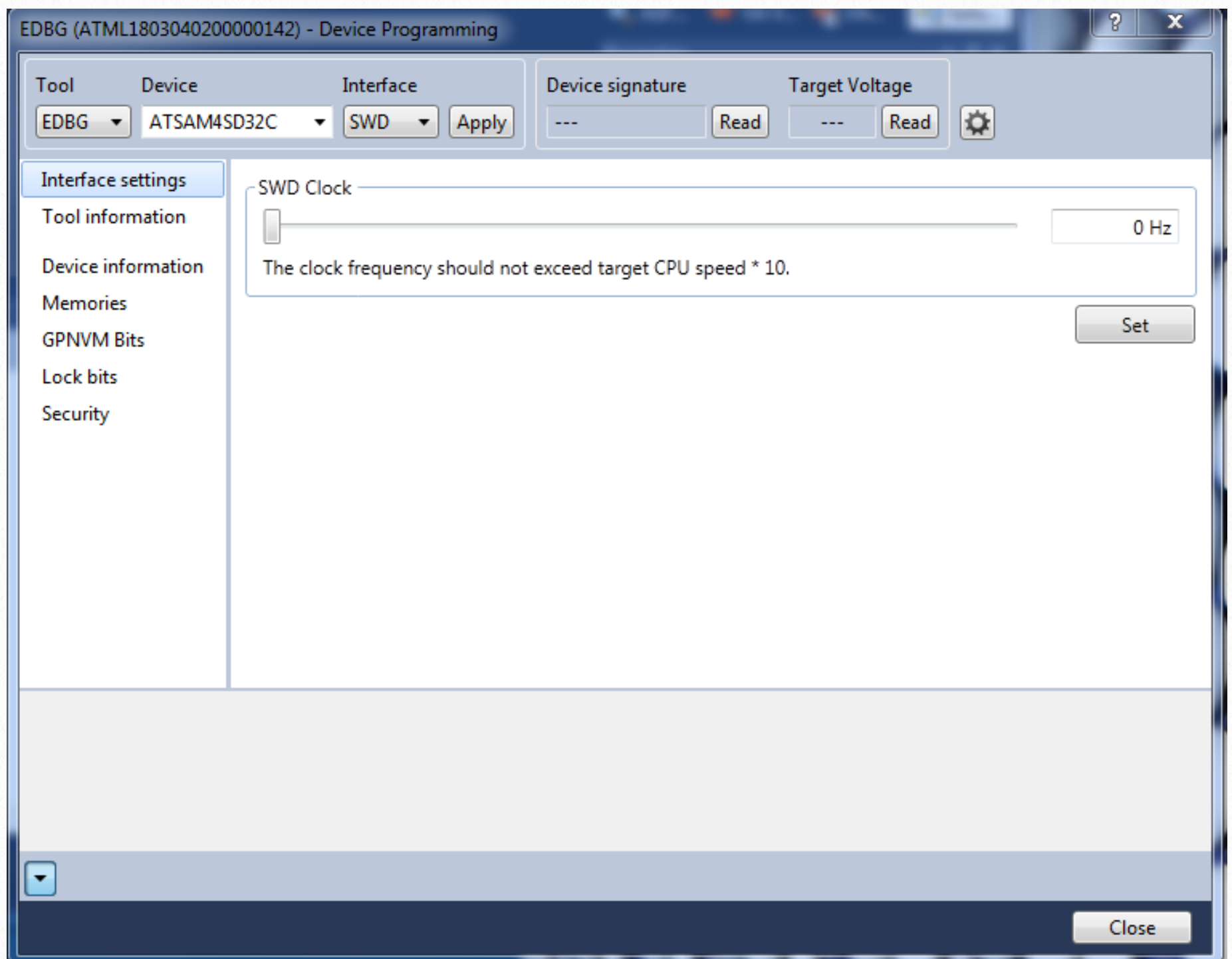
If you do not see the EDBG drop down option, your EDBG driver installation is probably incorrect during the Atmel Studio installation. I had a problem that I was using Atmel Studio 6.1 for developing Hempstick, when v.6.2 came out supporting more boards, Atmel said that 6.1 and 6.2 can co-exist on the same system.... Not true... at least the USB driver for

EDBG does not work right when both 6.1 and 6.2 are installed on the same system. It not only does not work in 6.2, it also rendered 6.1's EDBG non-functional. I had to manually uninstall everything, including 6.2, 6.1 and all the Atmel Tool Chain, USB drivers, and ASF, everything Atmel, reboot and reinstall only 6.2.

Try that if you have problem completing the following steps.

Now, select [EDBG], and then it should automatically populate the Device field and Interface field. If not, select them as what the previous screenshot shows.

Then, click the [Apply] button. And you should see the SWD Clock slider showing up.



Sometimes, when you click on the [Apply] button, it might detect that your EDBG chip has

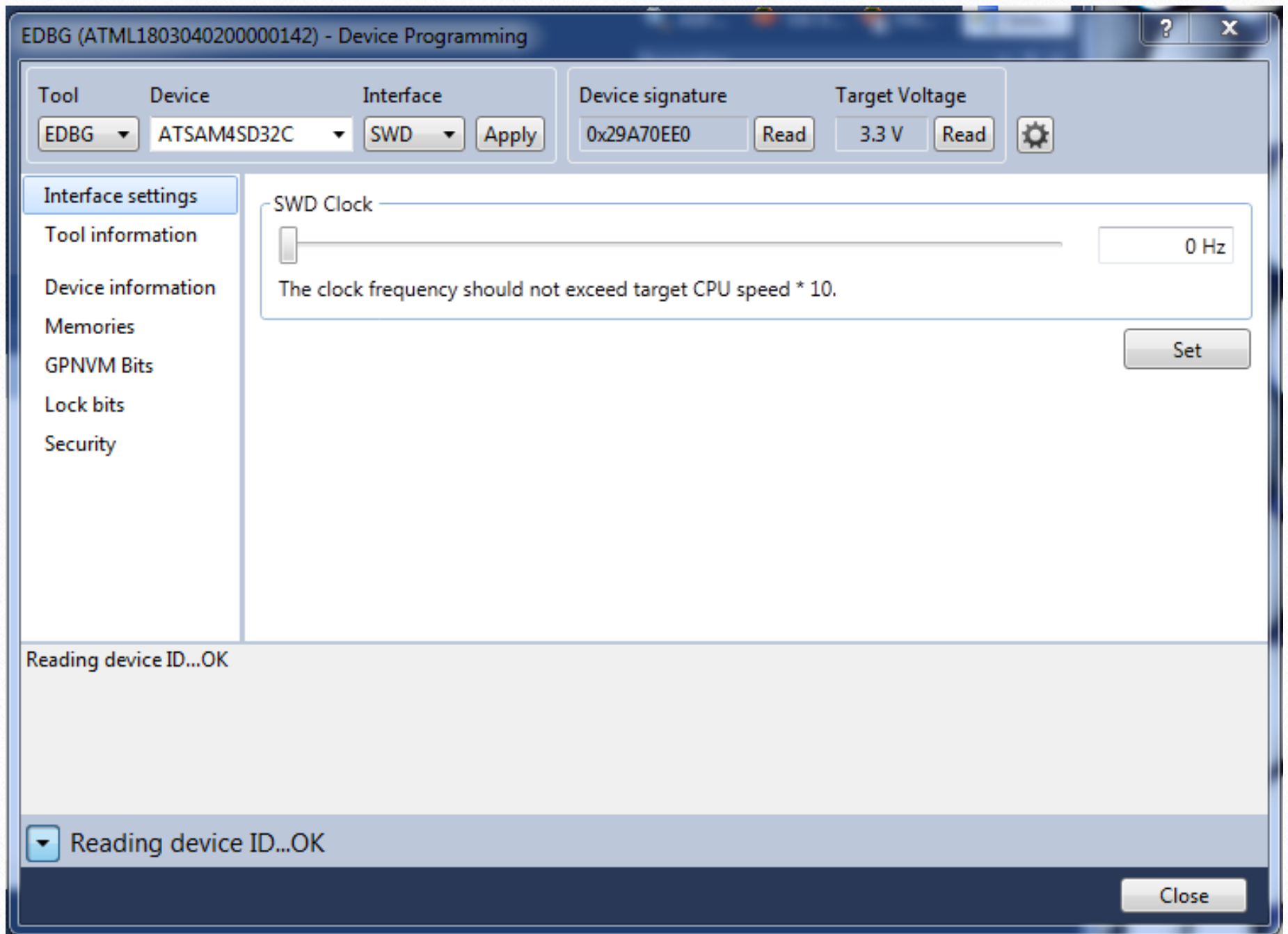
an older version of firmware, and you must update the firmware before you can continue (Atmel Studio will automatically guide you through the EDBG firmware update process). Atmel Studio always ship with newest firmware for every debugger firmware they support, including the EDBG. Atmel Studio must operate with the same version of firmware as what's on the debugger, and that would be the latest version! Let it update the firmware!

However, if you are running the Windows under VMware, like I do. This update of firmware might not work. What the update EDBG firmware process does is that Atmel Studio will instruct the EDBG chip to go into a bootloader mode, using the same VID/PID and then send over the new firmware for update. Unfortunately, VMWare under a Mac that I use is not able to detect that change when it still uses the same VID/PID. This would cause the EDBG USB driver to go funky. Atmel Studio will find the update of firmware failed because it is not able to detect the bootloader device, while the EDBG USB driver, I guess, keeps waiting for a bootloader device to show up. Thus, the EDBG USB driver is completely unusable at the point. And you would have to reboot the Windows VM.

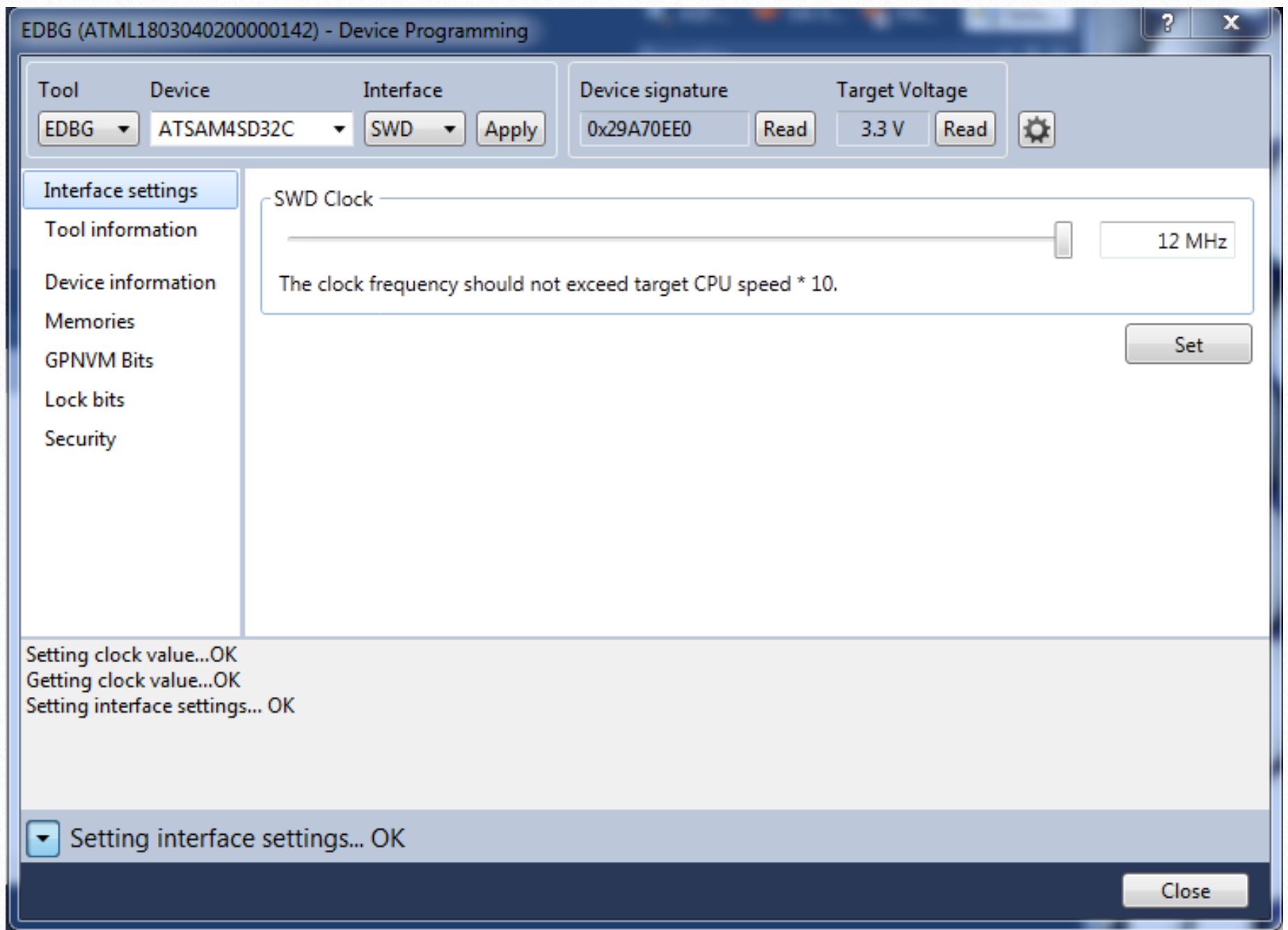
The only workaround I know for this problem is to update the firmware on a physical Windows machine and then take the updated board back to the development VM.

Why developing a Windows thing on a VM under a Mac? See, the Mac is my main desktop, and I run multiple Windows VMs (and Linux too) under it. One Windows VM has all the development tools, while another Windows VM serves as a target machine that has absolutely no contamination from the development tools! So, for the debugger USB I instruct VMWare to connect it to the development VM, while for the SAM4S USB, I instruct VMWare to connect to the target VM. This way, not only the target VM is uncontaminated to ensure that it works with a vanilla Windows installation, if for some reason the USB driver (if I ever write one) messes up the target VM and it no longer boots, I can easily whack the VM and reinstall, without having to install all the development tools! You have to understand that setting up a development machine takes a long time, installing 20/30 different development tools is very common and you have to make sure they are all configured and working correctly. I am not going to put an under development buggy kernel mode driver on such a development machine!!!

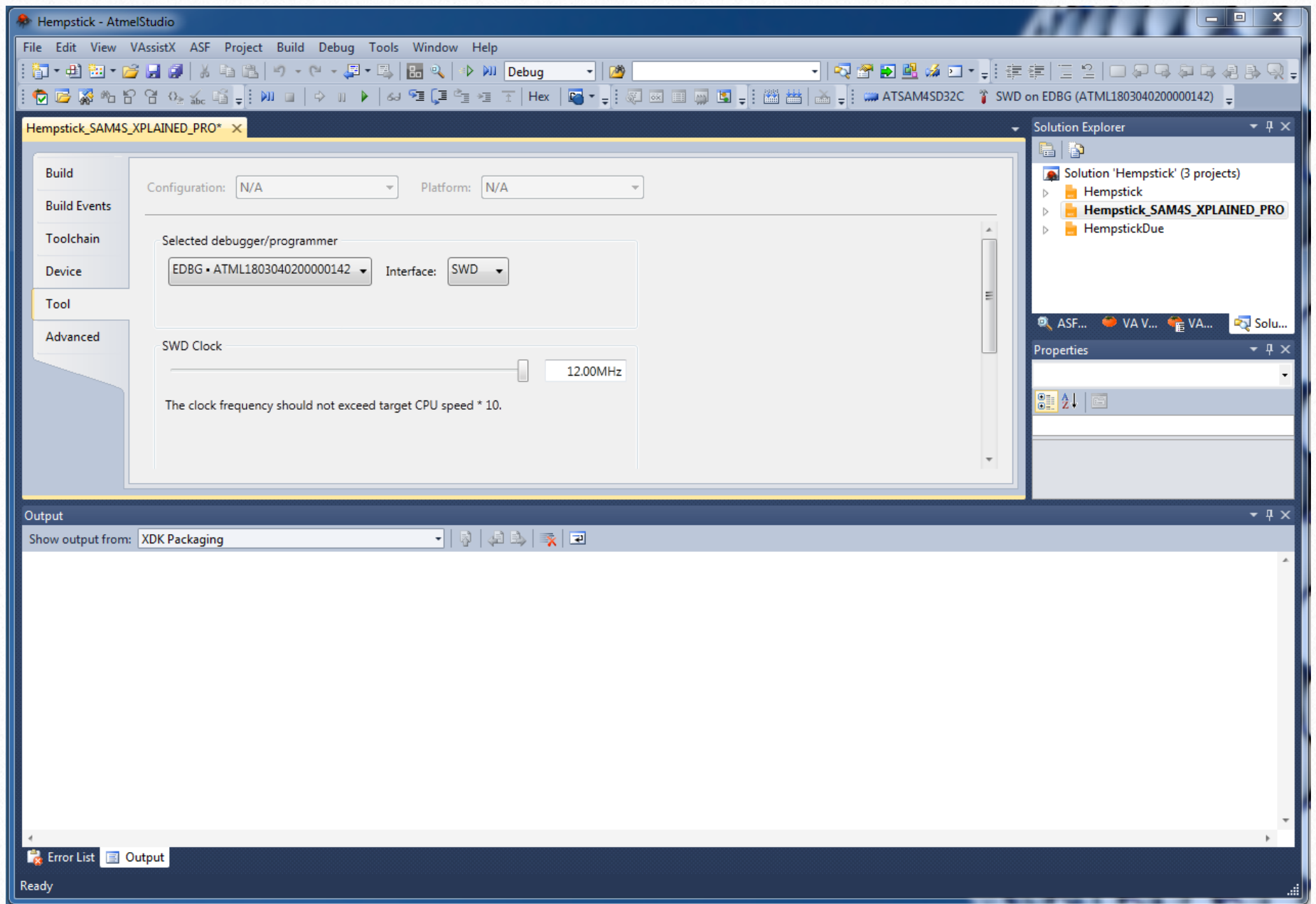
Now, click on the upper right [Read] button. You should now see the Device signature gets populated. This is the unique chip Id read back from the SAM4S chip via the EDBG channel. And you should also see the Target Voltage read as 3.3V. Now we have just verified that your EDBG chip and its USB driver works.



While you are here, slide that SWD Clock slider all the way to the right and click [Set], please. And close this window.



Go to the Solution Explorer again, select the Hempstick_SAM4S_XPLAIN_Pro project again, [R-Click] -> [Properties]. And you should see the following screen. Select the [Tool] tab on the left if you have to. You should see the [Selected Debugging Interface] set to EDBG..., [Interface] set to SWD, and the [SWD Clock] set to 12MHz. If they are not so, set them so and save the project with [Ctrl-S].



We are done verifying the Atmel Studio and the EDBG! Now we can finally go do the real work of making a rudder controller!!!

3

Configure & Build a Custom Hempstick Rudder Controller

Hempstick comes with several projects inside the Hempstick.atsln “solution.” A solution is a Microsoft Visual Studio way of organizing several related projects. Each project contains the complete collection of files to product a final “artifact.” In the Hempstick solution, there are initially 3 projects.

- I. Hempstick
- II. Hempstick_SAM4S_XPLAIN_Pro
- III. Hempstick_Arduino_Due

The Hempstick project does not support any board. It is a generic Hempstick project other board-specific projects are based on. You need not concern yourself with this project unless you are trying to support additional boards.

In this demo rudder project, we will be using the Hempstick_SAM4S_XPLAIN_Pro project. From here on, we will only use files under this project. So, any references to file names are under this project, unless we specifically indicate otherwise.

The Hempstick_SAM4S_XPLAIN_Pro project, by default, comes with the following features.

- 64 buttons.
- 8 axes.
- The first 24 buttons are configured to read a TM Stick (Cougar or Warthog sticks).

- Runs at 120MHz.
- USB report rate is 1,000Hz (the max. a full -speed USB connection can do.)

Yes, the SAM4S chip is capable of High-Speed USB enabling even higher report rate, but for a joystick or rudder, you really don't need anything higher than that. Humans can only, on average, do about 200ms response. That is, starting you see something to react to it with muscle movements, it takes about 200ms. We are already reporting 200 times more than that. Sure, we do need more than 5 reports per second to “predict” in between actions. But think about it this way. Your screen refresh rate is probably going to top out at 120Hz, but we are already reporting more than 8 times of that. You don't need anything higher than that, period!

However, it would benefit from even higher USB report rate if in the future we hook up Hempstick to Ethernet and other more advanced machine processing, but not now for a lowly rudder or controlling some instrument panels.

What We Want to End Up With

We just need 3 axes, find the three pins and connect them to 3 potentiometers. That's it.

What about the built-in 64 buttons and the other 5 additional axes? Who cares? We don't connect them. The host computer is still going to get reports about their values, but they are full of junk data. All buttons will get “not-pressed” report values. And the other 5 axis reports will get junk data in there. Do you care? I don't. just don't configure those extra buttons and axes in your sim and you are good.

To really get rid of the unneeded 5 axes and 64 buttons, it would require some more advanced customizations, like changing the USB HID report descriptor. It's a little bit more advanced topic than this quick rudder project is about. We will get into that some other time.

However, we will go into what are needed to change your VID/PID, and the manufacturer and product name. And we will show you how to verify the functionality with hardware pots, and then in Windows.

Modify Top Level USB Controller Information

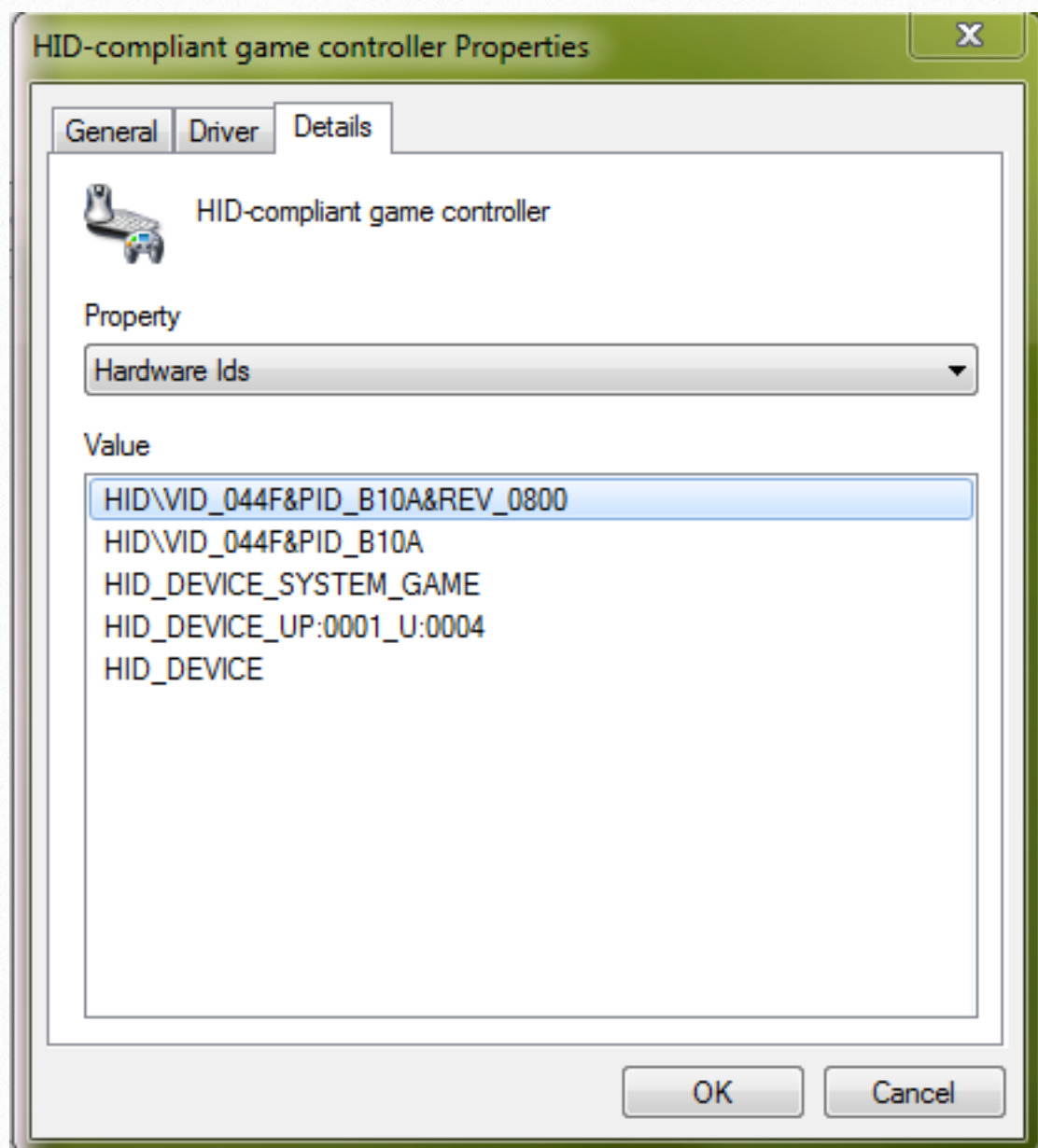
Modify VID/PID

Open the src\config\conf_usb.h file, and look for the following two lines.

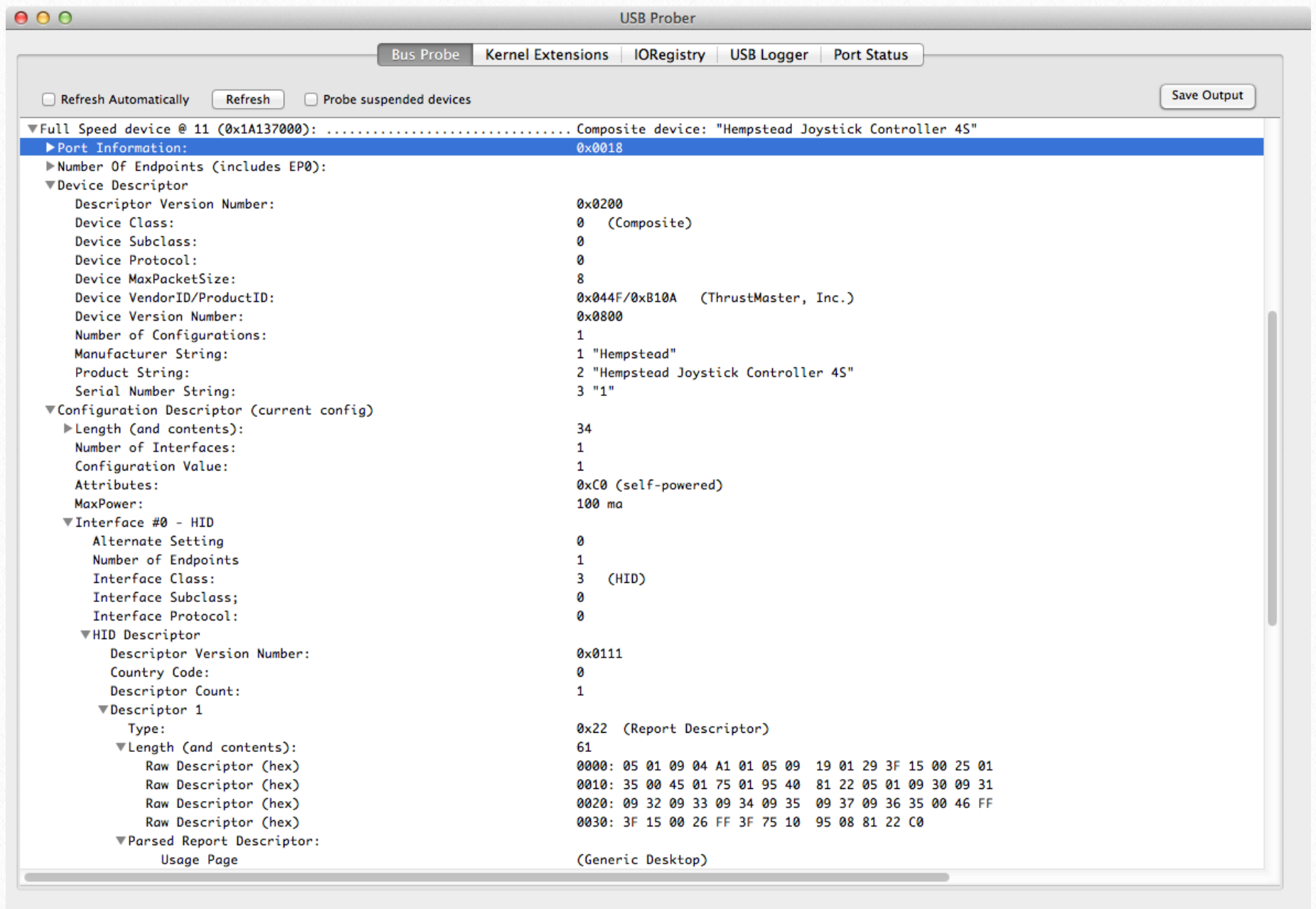
```
#define USB_DEVICE_VENDOR_ID      0x44F
#define USB_DEVICE_PRODUCT_ID     0xB10A
```

The Vendor Id 0x44F is ThrustMaster's VID. 0xB10A is T.16000M's PID. You need to change these to what you want and be careful not to collide with other USB devices' VIDs/PIDs in the system.

The easiest way for you to find a valid VID/PID is to... use other Vendor's VID/PID! Plug in a device you wish to "masquerade" as into your Windows OS. Then go to [Start] -> [Devices & Printers] , select the newly plugged in USB controller [R-Click] -> [Properties] -> [Details] tab -> [Hardware Ids] on the Property drop down. And you should see something similar to the following.



If you have a Mac. That's even better. Go look for an “deprecated” application from Apple called USB Prober.app, and you'd get something similar to the following.



See? It not only gives you VID/PID, it even gives you the USB descriptors, including both binary & parsed version. This is very useful when we get into the more advanced topic of modifying the USB descriptors in Hempstick.

Modify Manufacturer & Product Name

In the same config\conf_usb.h file, find the following lines.

```
#define USB_DEVICE_MANUFACTURE_NAME    "Hempstead"
#define USB_DEVICE_PRODUCT_NAME        "Hempstead Joystick Controller 4S"
#define USB_DEVICE_SERIAL_NAME         "1"
```


Change them to whatever you wish. Put your Call Sign on it! Why would you want to use my call sign???!?

I would also advise that you keep the line `USB_DEVICE_SERIAL_NAME "1"` there. Putting this serial number there allows you to plug the controller into one USB port and then the next time you plug it into another USB port and Windows OS will remember that same setting for the same device instead of treating it as a different USB device.

Configure the ADC & Buttons

What to configure? Nothing really!

The default Hempstick SAM4S XPLAIN Pro project gives you 64 buttons and 8 axes. All 8 axes are assigned correctly to certain ADC pins, but not all buttons are assigned. In fact, the default setting turns on the TMStick module so the first 24 USB buttons are automatically assigned to various TMStick buttons.

We need to at least turn off the TMStick module. And if desirable, find the ADC pins we wish to use and reassign them. Technically, you could just use the default ADC pin assignment and wire your pots accordingly, but we are going to demonstrate how to re-map these ADC pins to different USB axes.

Turn on/off the TMStick

Open the file `src\conf\config_hempstead.h` and find the following line.

```
#define CONF_ENABLE_TM_STICK_IN_BUTTON 1
```

Change it to 0 and save the file.

```
#define CONF_ENABLE_TM_STICK_IN_BUTTON 0
```


That's it. Technically, you are ready to burn the Hempstick firmware.

But, it's worthwhile to explain a little bit more in case you wish to remap the ADC pin to USB axes mapping, even though we have explained this in the Hempstick User's Guide document already. We never explained where to find these information. We will do it here.

Re-Assign ADC Pins

Let's take a peek at the SAM4S specification sheet, just a page.

If you take a look at the PA17 row in the next screenshot, you'd find that it contains multiple peripheral functions. There are so many functionalities implemented on the SAM4S chip, it's impractical to route all of them out to each pin. I mean, the SAM4S chip version we use already has 100 pins, if Atmel route every peripheral function out to dedicated individual pins, the chip would probably have something like at least 300 pins. Pins are expensive to route out and you probably don't need all of them all at once. So, Atmel "multiplex" the pins/functions. This is a very common practice in the industry, not just Atmel doing it.

The PA17 pin has 5 functions. They are:

- TD
- PCK1
- PWMH3
- AD0
- Unlisted GPIO

We are interested in the AD0 function. This is the channel 0 of the on die 12 bit ADC. This would be PA17 pin's "Extra Function." Extra functions in Atmel chips are usually configured at reset (reboot) automatically, so there is no need to explicitly configure it. But, if we wish to use the PA17 pin not for ADC but for its PWMH3 function, then we need to explicitly configure the PA17 pin to use Peripheral C function. Note that GPIO function is not listed in the table. But most of pins can be configured as GPIO (as Hempstick buttons).

11.2.1 PIO Controller A Multiplexing

Table 11-2. Multiplexing on PIO Controller A (PIOA)

I/O Line	Peripheral A	Peripheral B	Peripheral C	Extra Function	System Function	Comments
PA0	PWMH0	TIOA0	A17	WKUP0		
PA1	PWMH1	TIOB0	A18	WKUP1		
PA2	PWMH2	SCK0	DATRG	WKUP2		
PA3	TWD0	NPCS3				
PA4	TWCK0	TCLK0		WKUP3		
PA5	RXD0	NPCS3		WKUP4		
PA6	TXD0	PCK0				
PA7	RTS0	PWMH3			XIN32	
PA8	CTS0	ADTRG		WKUP5	XOUT32	
PA9	URXD0	NPCS1	PWMFIO	WKUP6		
PA10	UTXD0	NPCS2				
PA11	NPCS0	PWMH0		WKUP7		
PA12	MISO	PWMH1				
PA13	MOSI	PWMH2				
PA14	SPCK	PWMH3		WKUP8		
PA15	TF	TIOA1	PWML3	WKUP14/PIODCEN1		
PA16	TK	TIOB1	PWML2	WKUP15/PIODCEN2		
PA17	TD	PCK1	PWMH3	AD0		
PA18	RD	PCK2	A14	AD1		
PA19	RK	PWML0	A15	AD2/WKUP9		
PA20	RF	PWML1	A16	AD3/WKUP10		
PA21	RXD1	PCK1		AD8		64/100 pins versions
PA22	TXD1	NPCS3	NCS2	AD9		64/100 pins versions
PA23	SCK1	PWMH0	A19	PIODCCLK		64/100 pins versions
PA24	RTS1	PWMH1	A20	PIODC0		64/100 pins versions
PA25	CTS1	PWMH2	A23	PIODC1		64/100 pins versions
PA26	DCD1	TIOA2	MCDA2	PIODC2		64/100 pins versions
PA27	DTR1	TIOB2	MCDA3	PIODC3		64/100 pins versions
PA28	DSR1	TCLK1	MCCDA	PIODC4		64/100 pins versions
PA29	RI1	TCLK2	MCCK	PIODC5		64/100 pins versions
PA30	PWML2	NPCS2	MCDA0	WKUP11/PIODC6		64/100 pins versions
PA31	NPCS1	PCK2	MCDA1	PIODC7		64/100 pins versions

11.2.2 PIO Controller B Multiplexing

Now, let's take a look at the User's Guide for the SAM4S XPLAINED Pro board. This page shows you on the SAM4S XPLAINED Pro board, the pin functions of the Ext 1 connector.

4. Hardware user guide

4.1 Connectors

This chapter describes the implementation of the relevant connectors and headers on SAM4S Xplained Pro and their connection to the ATSAM4SD32C. The tables of connections in this chapter also describes which signals are shared between the headers and on-board functionality.

4.1.1 I/O extension headers

The SAM4S Xplained Pro headers EXT1, EXT2 and EXT3 offers access to the I/O of the microcontroller in order to expand the board e.g. by connecting extensions to the board. These headers all comply with the standard extension header specified in [Xplained Pro Standard Extension Header](#). All headers have a pitch of 2.54 mm.

Table 4.1. Extension header EXT1

Pin on EXT1	SAM4S pin	Function	Shared functionality
1	-	-	Communication line to ID chip on extension board.
2	-	-	GND
3	PA17	AD[0]	
4	PA18	AD[1]	
5	PA24	GPIO	PIOD Interface Header
6	PA25	GPIO	PIOD Interface Header
7	PA23	PWMH0	PIOD Interface Header
8	PA19	PWML0	
9	PA1	WKUP1/GPIO	
10	PA6	GPIO	DGI_GPIO0 on EDBG
11	PA3	TWD0	EXT2 and EDBG
12	PA4	TWCK0	EXT2 and EDBG
13	PA21	USART1/RXD1	EXT2
14	PA22	USART1/TXD1	EXT2
15	PA11	SPI/NPCS[0]	
16	PA13	SPI/MOSI	EXT2, EXT3, LCD connector (EXT4) and EDBG
17	PA12	SPI/MISO	EXT2, EXT3, LCD connector (EXT4) and EDBG
18	PA14	SPI/SPCK	EXT2, EXT3, LCD connector (EXT4) and EDBG
19	-	-	GND
20	-	-	VCC

Table 4.2. Extension header EXT2

Pin on EXT2	SAM4S pin	Function	Shared functionality
1	-	-	Communication line to ID chip on extension board.
2	-	-	GND
3	PB0	AD[4]	
4	PB1	AD[5]	
5	PC24	GPIO	DGI_GPIO2 on EDBG
6	PC25	GPIO	DGI_GPIO3 on EDBG
7	PC19	PWMH1	

From this table, we see that Ext1:3 is the ADC[0] for PA17 pin. We need three of these, let's pick PA17, PA18, and PB0. So, we want it to end up like this.

- PA17 (ADC0) maps to Z (main rudder axis)
- PA18 (ADC1) maps Rx (left toe brake)
- PB0 (ADC4) maps Ry (right toe break).

Now, open src\conf\config_usb.c file and find the following entry.

```
//! HID report descriptor for standard HID mouse
UDC_DESC_STORAGE udi_hid_joystick_report_desc_t udi_hid_joystick_report_desc = {
    {
        0x05, 0x01,          // USAGE_PAGE (Generic Desktop)
        0x09, 0x04,          // USAGE (Joystick)
        0xa1, 0x01,          // COLLECTION (Application)
        0x05, 0x09,          //  USAGE_PAGE (Button)
        0x19, 0x01,          //  USAGE_MINIMUM (Button 1)
        0x29, 0x3f,          //  USAGE_MAXIMUM (Button 63)
        0x15, 0x00,          //  LOGICAL_MINIMUM (0)
        0x25, 0x01,          //  LOGICAL_MAXIMUM (1)
        0x35, 0x00,          //  PHYSICAL_MINIMUM (0)
        0x45, 0x01,          //  PHYSICAL_MAXIMUM (1)
        0x75, 0x01,          //  REPORT_SIZE (1)
        0x95, 0x40,          //  REPORT_COUNT (64)
        0x81, 0x22,          //  INPUT (Data,Var,Abs,NPrf)
        0x05, 0x01,          //  USAGE_PAGE (Generic Desktop)
        0x09, 0x30,          //  USAGE (X)
        0x09, 0x31,          //  USAGE (Y)
        0x09, 0x32,          //  USAGE (Z)
        0x09, 0x33,          //  USAGE (Rx)
        0x09, 0x34,          //  USAGE (Ry)
        0x09, 0x35,          //  USAGE (Rz)
        0x09, 0x37,          //  USAGE (Dial)
        0x09, 0x36,          //  USAGE (Slider)
        0x35, 0x00,          //  PHYSICAL_MINIMUM (0)
        0x46, 0xff, 0x3f,    //  PHYSICAL_MAXIMUM (16383)
        0x15, 0x00,          //  LOGICAL_MINIMUM (0)
        0x26, 0xff, 0x3f,    //  LOGICAL_MAXIMUM (16383)
    }
}
```



```

0x75, 0x10,          // REPORT_SIZE (16)
0x95, 0x08,          // REPORT_COUNT (8)
0x81, 0x22,          // INPUT (Data,Var,Abs,NPrf)
0xc0                 // END_COLLECTION
    }
};

```

The above is the default USB HID report. Don't worry about all the things you don't know what they do. Just look at the order of the USAGE(X)... rows.

It says, the first axis, the 0 axis, is X, axis 1 is Y, etc. We want axes 3, 4, and 5.

So, we will have the followings.

- PA17 (ADC0) maps to Z (main rudder axis), USB axis 3
- PA18 (ADC1) maps Rx (left toe brake), USB axis 4
- PB0 (ADC4) maps Ry (right toe break), USB axis 5.

Now, let's go rearrange the axis mapping in Hempstick. Open the file `src\config\conf_hempstead.c` and find the following entries.

```

rtos_adc_data_type g_adc_data = {
    .data = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
#ifdef CONF_BOARD_SAM4S_XPLAIN_PRO
    .channel_flags = {ADC_CHANNEL_ENABLE_MASK, 0, 0, 0, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, 0, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, 0, 0, 0, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, 0},
#elif defined(CONF_BOARD_ARDUINO_DUE)
    .channel_flags = {0, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, ADC_CHANNEL_ENABLE_MASK, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
#endif
    .channel_mapping = {0, UINT8_MAX, UINT8_MAX, UINT8_MAX, 1, 2, UINT8_MAX, 3, 4, 5, UINT8_MAX, UINT8_MAX,
        UINT8_MAX, 6, 7, UINT8_MAX},
    .num_channel_enabled = 0,
    .adc_config = 0,
    .mutex = NULL,
    .rtos_task_semaphore = NULL,
    .pdc_sample_data = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};

```



```
};
```

Ignore the entries under `#elif defined(CONF_BOARD_ARDUINO_DUE)`. We are not using this board.

In the `.channle_flags = {.....}` array, this is the 0-based index for ADC channels. This array is used to tell Hempstick which ADC channels are enabled. The default enables the following 8 ADC channels.

0, 4, 5, 7, 8, 9, 13, 14

Why? That's the pins routed out on the SAM4S XPLAINED Pro boards. Note that, some of the pins are not marked as AD[?] on the board's User's Guide, but marked as something else, like the PB2 and PB3 pins are marked as USART1/TXD1 and USART1/TXD1. We don't use serial ports on Hempstick, so we reconfigure them as ADC pins.

Another one is the PA18 pin, which is the ADC1 channel. But we are not using it as an ADC pin, because this pin's another function, RD, which is crucial for running the SSC module to read the TMStick. So, we trade one ADC channel for the ability to read the TMStick. Ok, no use of the PA18 pin for ADC[1].

Hold on, we can't use PA18 pin for ADC? We need to change our mapping to the followings.

- PA17 (ADC0, Ext1:3) maps to Z (main rudder axis), USB axis 3
- PB0 (ADC4, Ext2:3) maps Ry (right toe break), USB axis 4.
- PB1 (ADC5, Ext2:4) maps Ry (right toe break), USB axis 5.

Now, look at the `.channel_mapping` array. We have the followings.

```
channel_mapping = {0, UINT8_MAX, UINT8_MAX, UINT8_MAX, 1, 2, UINT8_MAX, 3, 4, 5, UINT8_MAX, UINT8_MAX,
UINT8_MAX, 6, 7, UINT8_MAX},
```


The `UINT8_MAX` entries mean no mapping for those particular ADC channels. But, we find the default mapping actually maps ADC0 to axis 0, ADC4 to axis 1, and ADC5 to axis 2. That's not what we want! Let's correct them. We get the following.

```
.channel_mapping = {3, UINT8_MAX, UINT8_MAX, UINT8_MAX, 4, 5, UINT8_MAX, 0, 1, 2, UINT8_MAX, UINT8_MAX,
UINT8_MAX, 6, 7, UINT8_MAX},
```

This mechanism allows us to map any ADC channel to any USB axis. Very flexible, but also a bit complicated. You must make sure that no double mapping (no crash if you do, but one channel will disappear, but it could be useful if you want to use one pot linked to two USB axes, say one pot to control two throttles; as to why you would want to do that, none of my business!).

Why so complicated? Well, the plan is that in the future, I might implement Ethernet or USB storage so that it's possible to remap them while the joystick is running. Imagine you send an UDP datagram to the joystick or plug in an SD card with different configuration, and Hempstick remaps all the axes and buttons? Or, the Hempstick provides you with a web page for reconfiguring the axes and buttons mapping? The possibility is endless, but this mapping mechanism must be there in order to support them!

Software framework and library design is quite different from fixed application design. In a fix application design, it only has to work one and only one way. But in framework/library design, it has to work in many ways, in particularly it also has to work in ways you can't think of yet. We are providing flexibility for what I cannot think of yet, at the expense of a little complexity. It's a trade off. This is the most difficult part of software design -- how much complexity are you willing to trade for flexibility? Experiences count here. I often see good application programmers try their hands on library/framework design and come up with something awfully stiff, or excessively complicated design for no apparent reason! Ok, enough rant about my day job.

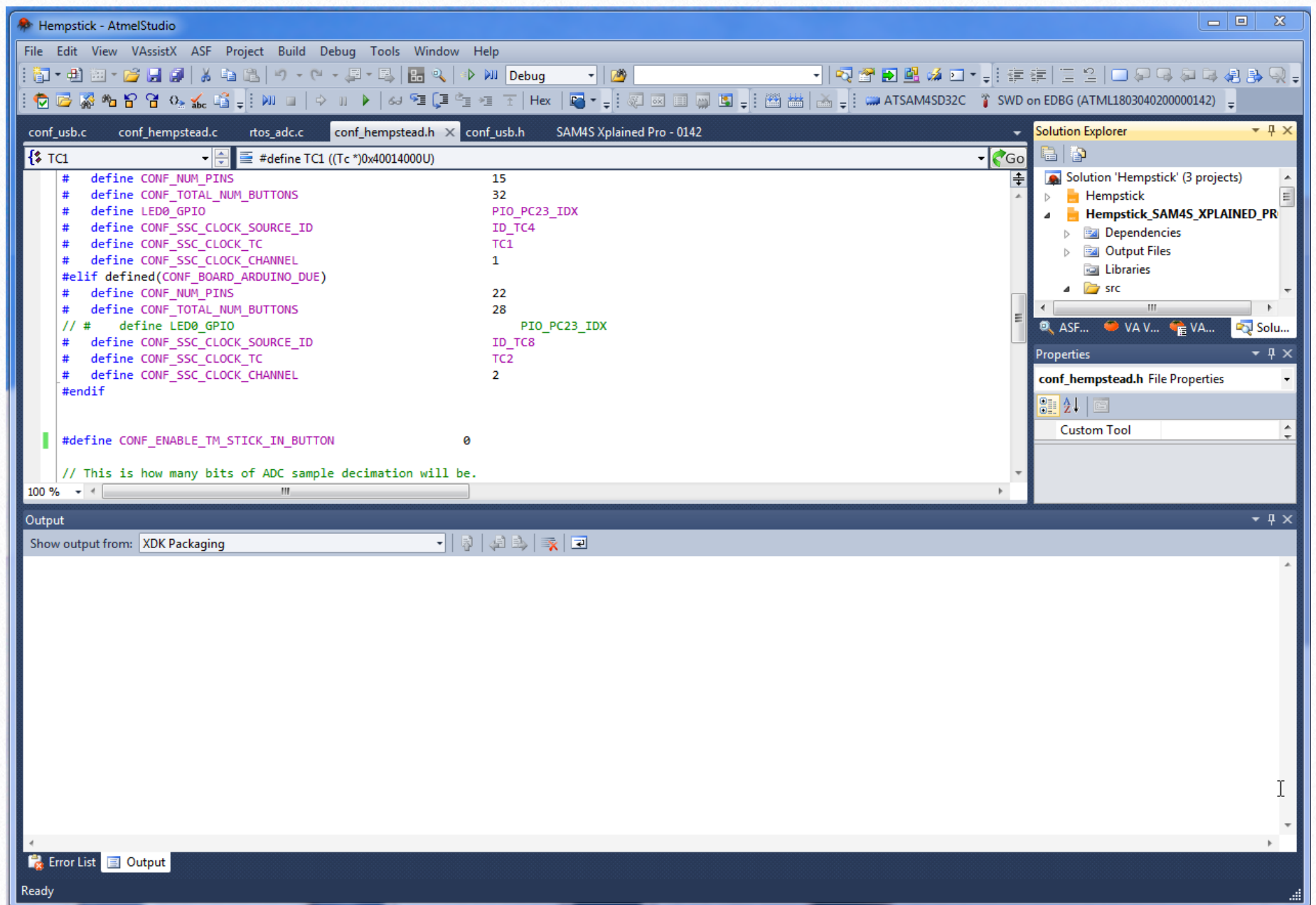
Let's BURN IT! BURN BABY BURN!


Burn the Firmware!!!

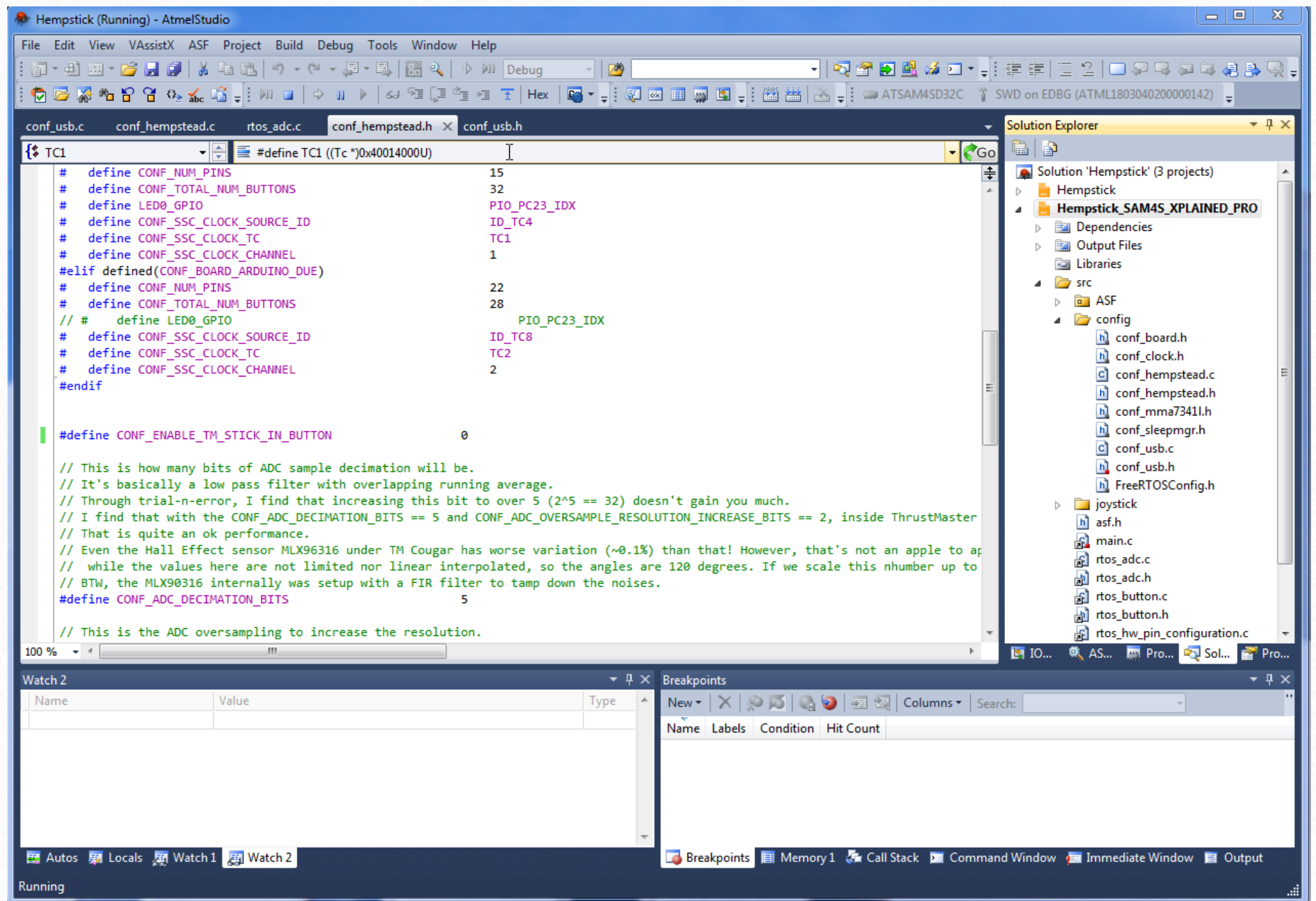
Technically, there are several steps.

1. Compile (or build)
2. Link the binary
3. Burn the binary into the SRAM of the SAM4S chip.

But, the Atmel Studio makes it easy for us. Just press the [Run] button, and it will do all of the above. Ok, ok, the button is really labelled “Start Debugging.”



Make sure the Hempstick_SAM4S_XPLAIN_Pro project is selected (and highlighted), then press the  button on the toolbar. If everything compile correctly, it should start uploading the build artifact (the binary) up to the chip. The compile log will have a lot of warnings... don't worry, they are normal.

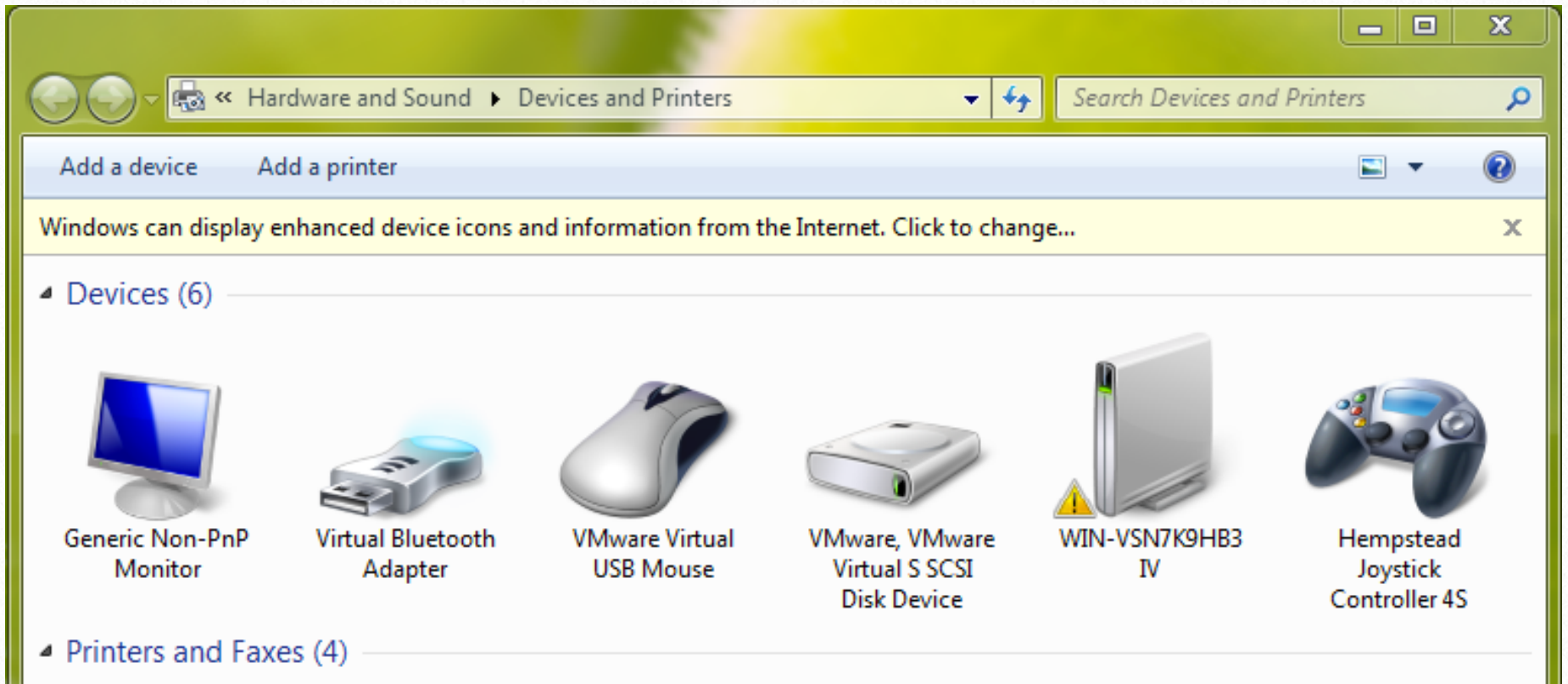



It should say “Running...” on the lower left corner status bar. If the burning is successful, it should also place the SAM4S chip under debugging mode. You are debugging it now.


If there are any error during the compilation, it will show you the errors without burning.

If you do not wish to place the SAM4S chip under debugging mode, then use the menu [Debug] -> [Start without debugging].

Now, plug in another micro USB cable between the SAM4S USB connector and the host computer. You should hear the Windows OS dinging you for the new Hempstick USB controller. Go to [Start] -> [Devices & Printers] and see if you have the new controller.

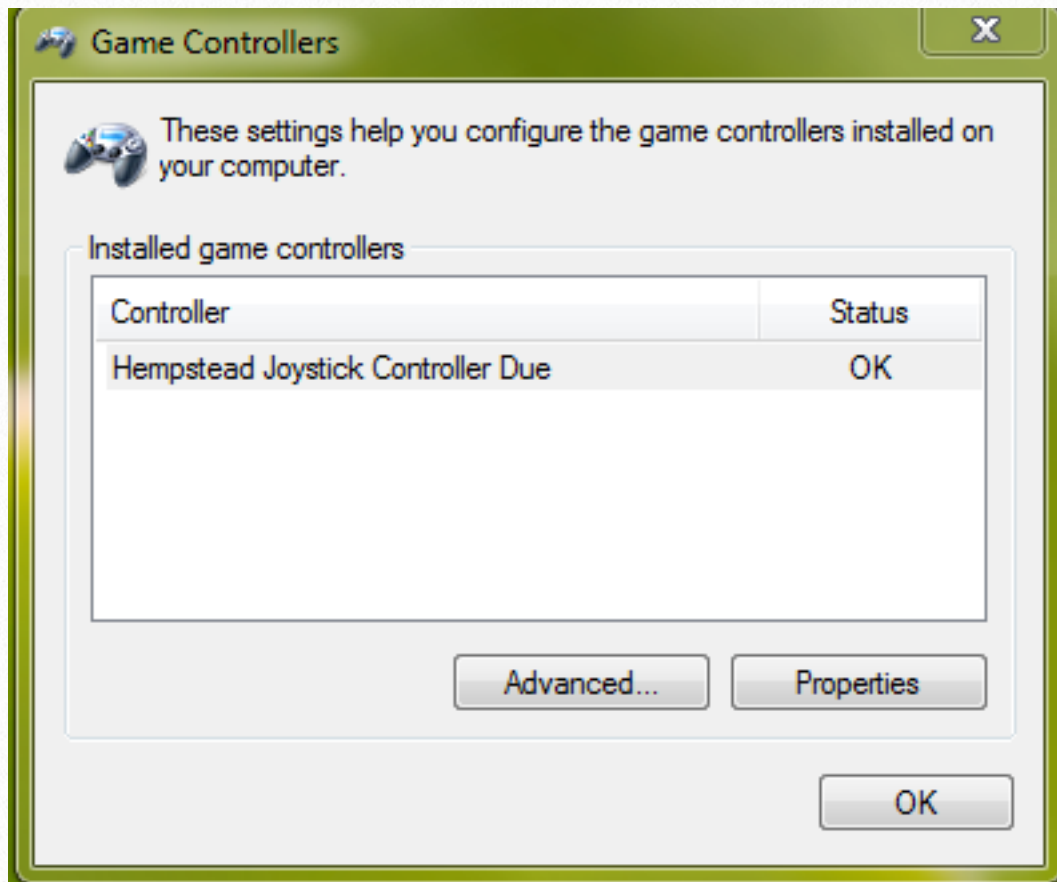


In addition, if you have changed any .h file, like `src\config\conf_hempstead.h`, I would advise instead of pressing just the  button, you do a two step “clean build” process as the followings.

1. Select the Hempstick SAM4S XPLAIN Pro project, in the menu [Build] -> [Rebuild Hempstick SAM4S XPLAIN Pro project].
2. If no error from step 1, press the  button.

This is because, sometimes the VisualStudio shell the Atmel Studio is built on does not detect the changes correctly and did not rebuild the affected header or .c files correctly. I am guessing it's because the “compiled header files” that are cached incorrectly. Do a clean build is the programmers’ “cure-it-all”, as opposed Windows users’ reboot.

Now, select the new Hempstick controller, [R-click] -> [Game Controller Settings], and you should see the following.



Wait!!!! Why does it say Hempstead Joystick Controller **Due**??? I said Hempstead Joystick Controller **4S**!!!

This is because, we used the same VID/PID, and the Windows OS internally caches this information as a device's "Friendly Name"! But, inside the Devices & Printers, it says "4S", and inside the Game Controllers, it says "Due!" Well, I know there is a way to change this... but it's a bit complicated, it requires registry editing... Stupid Windows!

If you figure out an easy way to correct this problem, please please please let me know!

Next up, hardware wiring.

4

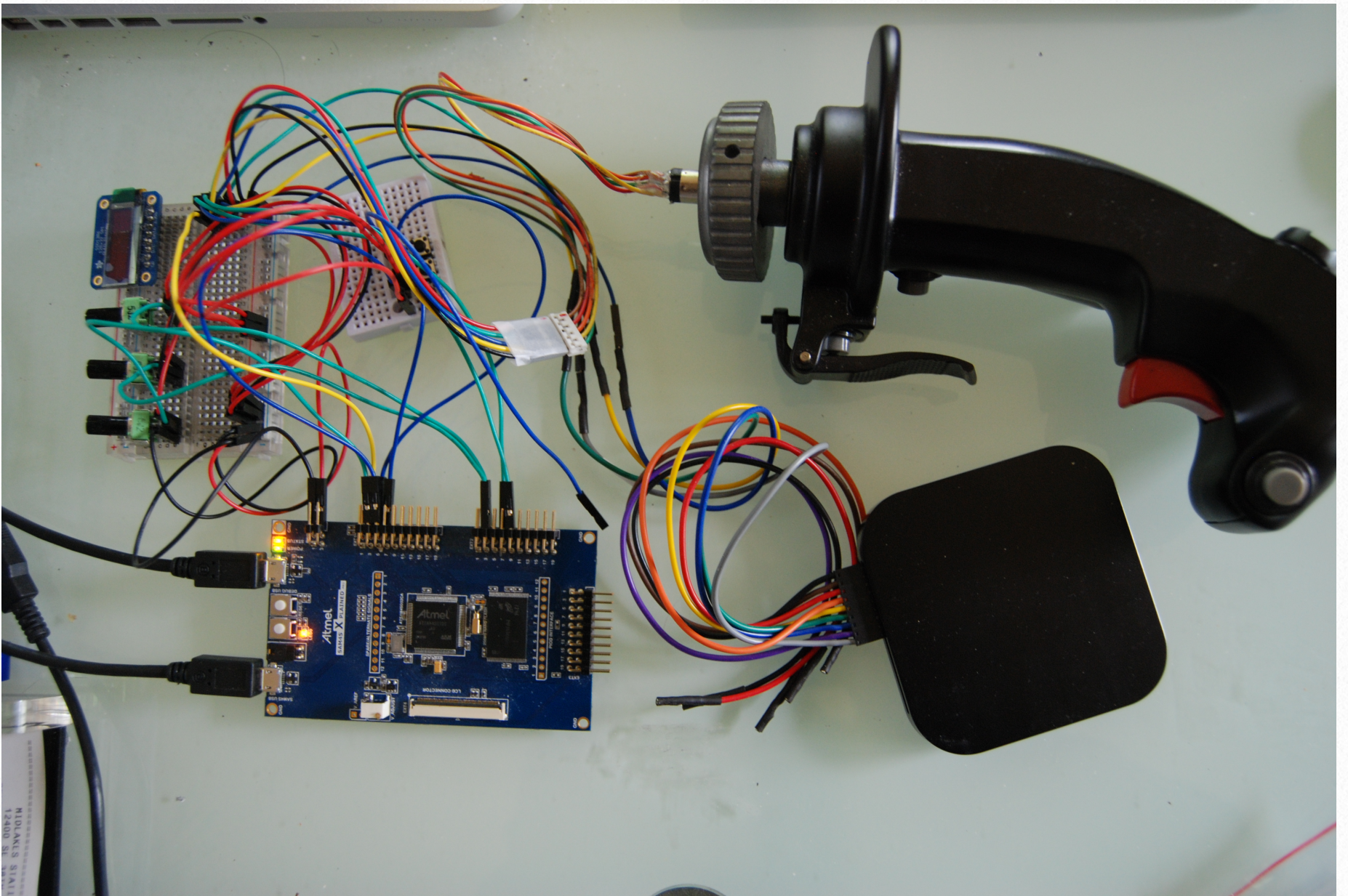
Wiring Up Hardware

I would highly recommend that you wire up some temporary pots using a prototyping breadboard, like this one, <https://www.sparkfun.com/products/9567>, to verify the functionality before you do permanent wiring to the physical rudder (or panels). While you are at it, also order some of these jumper wires, <https://www.sparkfun.com/products/9194>. They are very handy for quickly wiring up and rewiring some test circuits without any soldering. They save a lot of times. Imagine you read the wrong pin name when constructing the circuit and there are tens of wires in the circuit. The picture below is how my setup looks like.

From the picture, you can see that I have 3 potentiometers plugged into the larger breadboard, 3 buttons plugged into a smaller breadboard, a Warthog Stick plugged into the larger breadboard, and a Saleae Logic Analyzer also plugged into the larger breadboard, all without one single soldering joint.

With this setup, I can quickly rewire by just swapping the jumper wires, change the code, and verify whether I get it right or not without pulling out any tools like a soldering iron, plier, or crimper. No tools required, except my hands.

These breadboard & jumper wires are not good for high speed MCUs. Our SAM4S, 120MHz, is quite a high speed one. But, we are not hooking into the high speed part, we are only using one or two MHz against the TMStick. So, this kind of setup is perfectly fine for this purpose.



Mine has a little bit of spaghetti tangle of wires. But, your 3 pots don't need much. You don't even need a breadboard, although I highly recommend one. Without a breadboard, you would have to split the 3.3V Vcc line and the ground line into three each.

Each pot will need 3 wires, a 3.3V Vcc line, one signal output line, and a ground GND line. Each pot would have three leads for these. Usually, the middle lead is the output wiper line. The outside two are either Vcc or GND, depending on which direction you want the "up." Connect the pots in 3-wire voltage mode! No 2-wire current mode monkey business!

Each of the Atmel Extension header would have 1 Vcc pin and 2 GND pins. Whenever it's marked Vcc, it's a the 3.3V. There is also a 4 pin power header on the board. This one provides another 5V header for your 5V peripherals, DO NOT HOOK THIS 5V INTO YOUR POTS!

The Atmel SAM chips are 3.3V devices, and they are not 5V tolerant!!!

YOU WILL DAMAGE THE MCU IF YOU HOOK 5V INTO ANY PIN!!!

Pin Connections

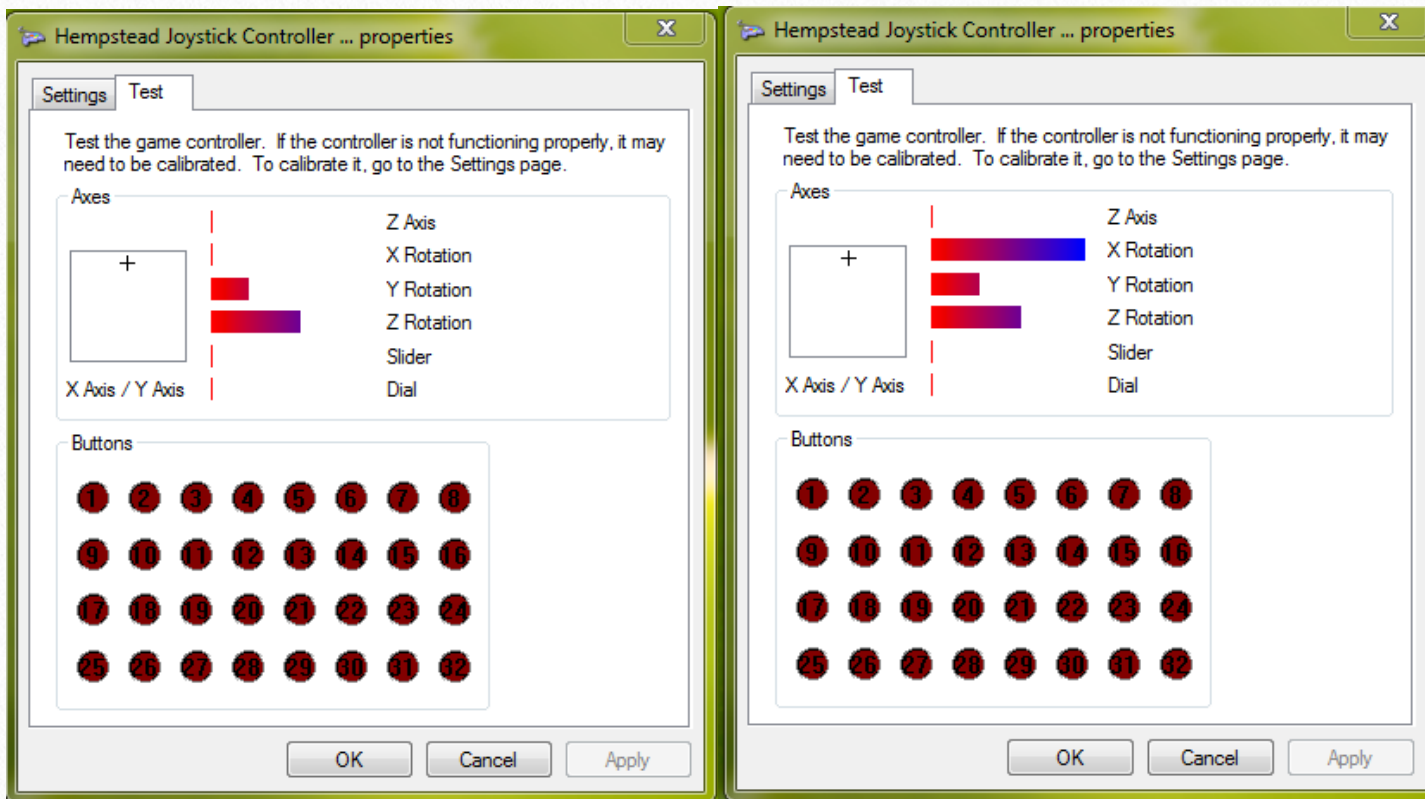
From the previous chapter, we know which pin should go where and where to find that information. Here's we will just list them in a table for the wiring job.

SAM4S XPLAINED Pro Board	SAM4S MCU Function	Rudder Pot
3.3V Vcc, Power Header Pin 4		All Pot Vcc
GND, Power Header Pin 2		All Pot GND
Ext 1, Pin 3	ADC 0	Main Pot Signal
Ext 2, Pin 3	ADC 4	Left Toe Brake Pot Signal
Ext 2, Pin 4	ADC 5	Right Toe Brake Pot Signal

Now, wire up your breadboard accordingly. Remember to stop the debugger, then unplug both the SAM4S and Debug USB cables from the SAM4S XPLAINED Pro board before you plug/unplug any wire!

I would advise that you do one thing at a time. Change one thing, verify it, then change another. This is a basic scientific method -- never change more than one variable so that if something goes wrong, you know that variable you just changed is the cause. It is also very applicable in software programming -- don't ever change many things and get into a tangle of mess you don't know which change you made caused the problem! It's so obvious, but believe me you that a lot of the younger programmers that work in my day job projects don't get that. And we are talking about "professionals", not some amateurs hacking away at home.

Wire one pot, 3 wires only, and verify it by plugging in only the SAM4S USB cable (no debugging this time). You should hear the ding from the Windows OS. Then, go to the [Devices & Printers] -> ... -> [Properties] to see the axis value by turning the potentiometer to see if the value get updated correctly.

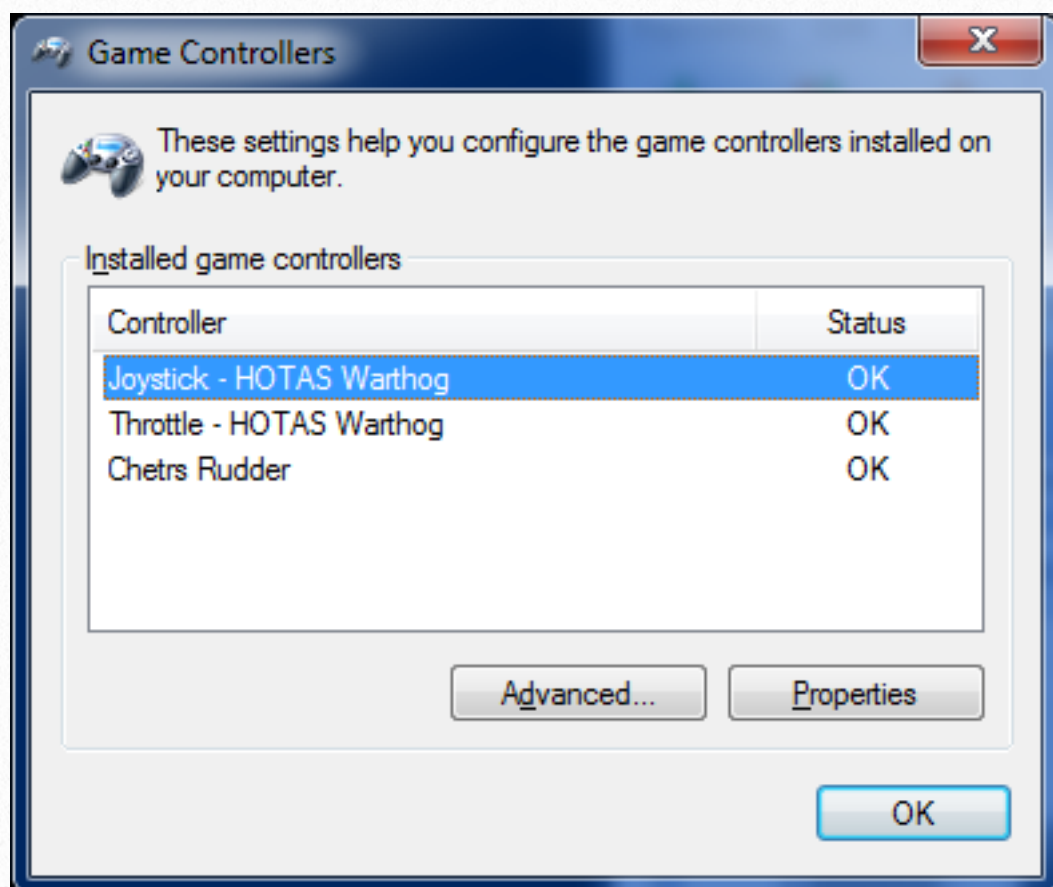


Did it update the right axis? Did it rotate in the desired direction?

For swapping the signal wires of pots... you know... you can re-wire them live, I won't tell. Once you verify that the wiring is correct for that one pot, label the wires! I don't care, Dymo, 3M tape, Sharpie, whatever. Just label them!

Keep adding more axes and verify all functions. Once you have verified these values, calibrate them in the Windows game controller window.

Again, my other computer with TARGET on it previously had a T.16000M plugged in and was called the Chetrs Rudder and Windows remembers that Friendly Name.

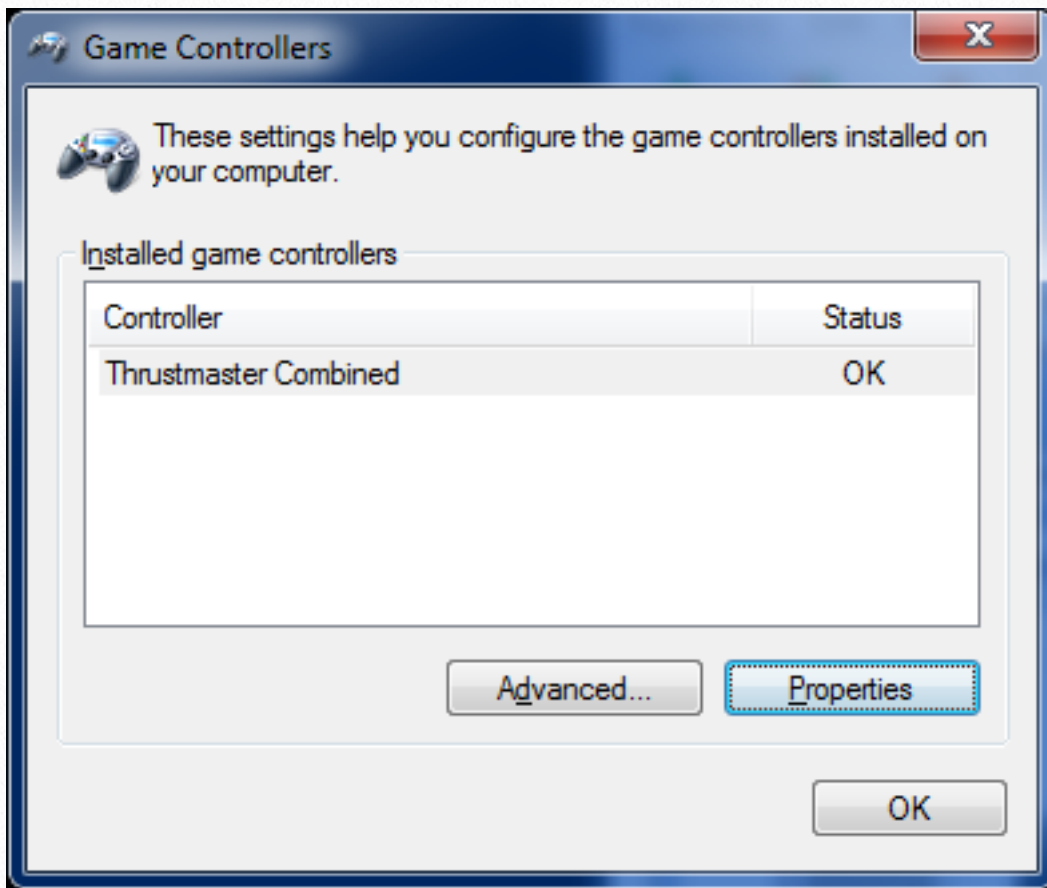


Now, launch TARGET and see if TARGET can see it. Again, I cannot tell you how, you will have to figure it out yourself. If you follow the instructions carefully, you should have figured it out by yourself already.

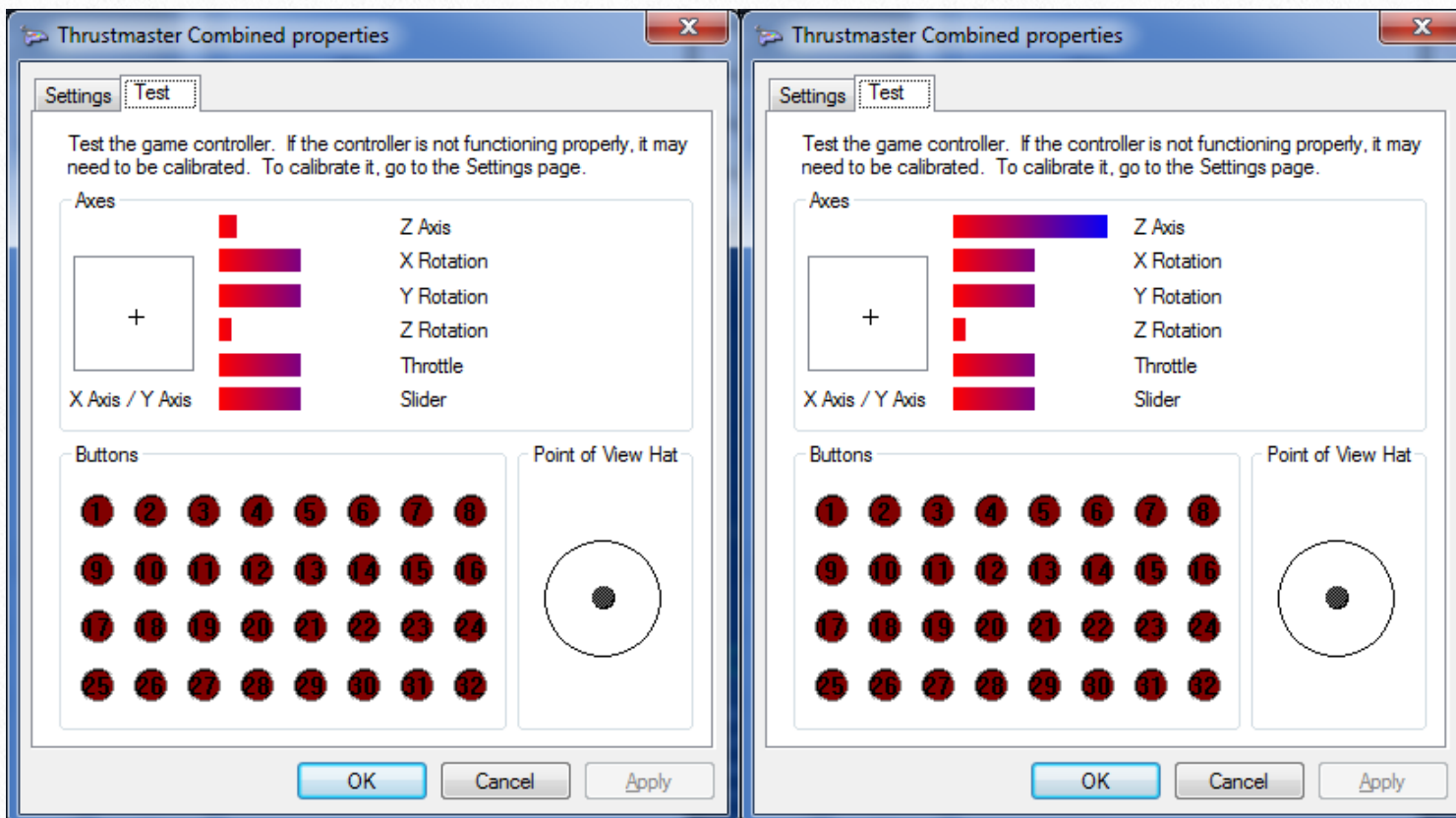


Now, if you can see the new rudder showing up inside TARGET GUI, you can program it inside TARGET and then run the TARGET script. You should then see in the Devices & Printers that every TM controllers you have programmed inside TARGET get unplugged programmatically and a virtual combined controller take their place. [R-Click] on this de-

vice -> [Game Controller Properties].



Now, rotate the pots and see if they change the values correctly on the correct axes.



You can now wire the SAM4S XPLAINED Pro board to the pots of your physical rudder.

But what to do with the physical rudder's original electronics???

DUMP IT!

I can't even find where my CH rudder's original main board is. I don't remember where I put it. I don't know, must be in one of the piles of electronics boards.

CAUTION!

If you intend to wire up 5V Hall Effect sensors, you must use at least a voltage divider to make them into 0 to 3.3V range before you wire them up to the Hempstick. Voltage dividers are not ideal, You are better off using an active Op Amp with some diode clamps to make sure they never exceed 3.3V output.

I will not go into that. it's your problem. Maybe search on SparkFun or AdaFruit and they might have a board just for that.

But I can tell you that Hall Sensors like MLX90316, even though it's an older generation of 5V Hall Effect Sensor, can be programmed to output only 3.3V by changing it's output curve and upper/lower clamping. So that it requires no voltage divider or any external active circuits to do the conversion from 5V to 3.3V. You basically feed it 5V power, but the output is programmed to 0-3.3V (or any range under 5V you program it to).

3.3V is the future in order to go high speed and cram more functions into the MCUs, convert or suffer the slow speed.

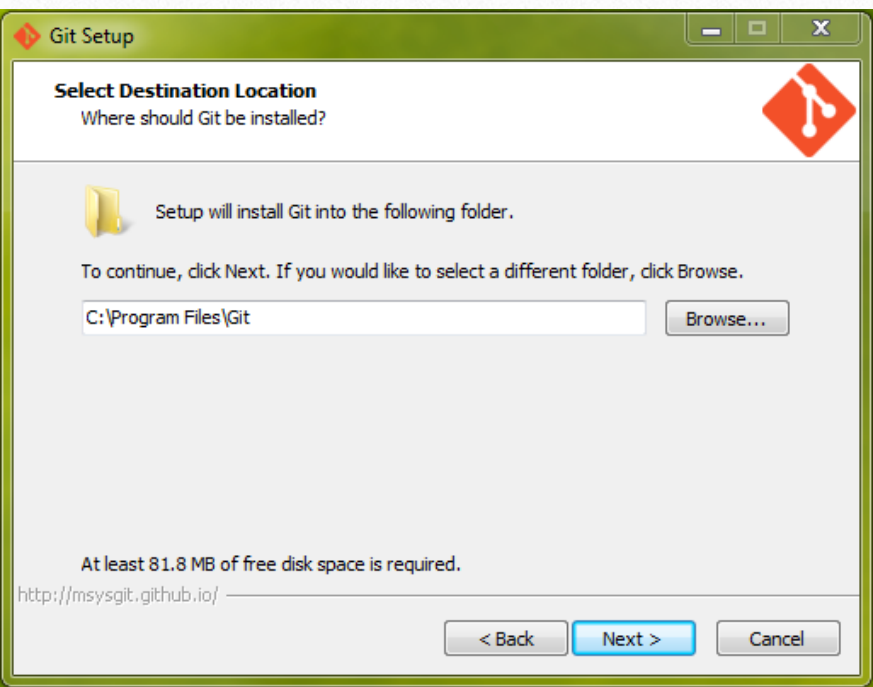
That's it! You have just made a custom Hempstick-based rudder controller.

A

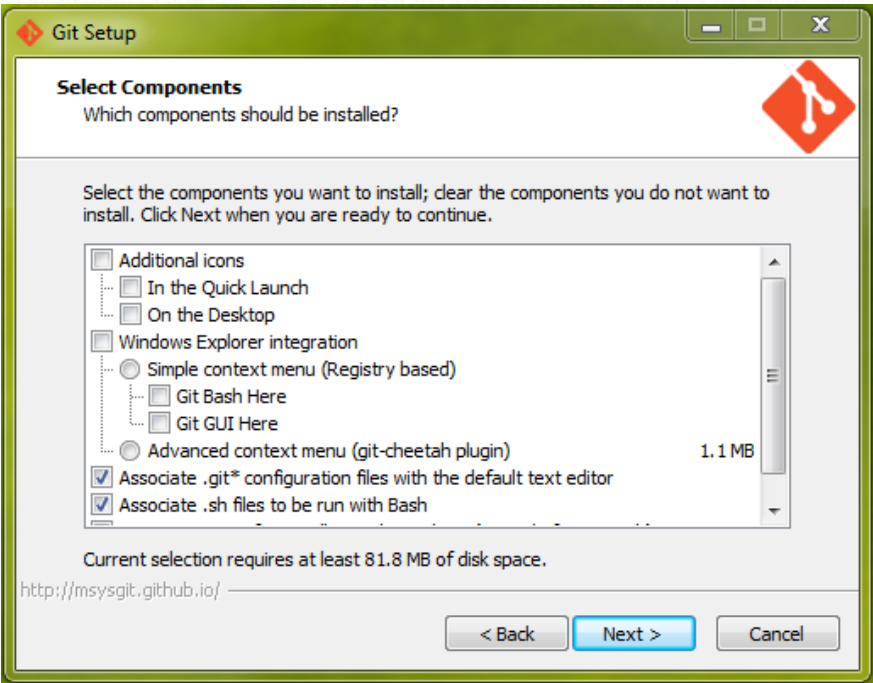
Installing MsysGit & SmartGit

When downloading SmartGit, if you don't have JRE (Java Runtime Environment installed, then please download the SmartGit version that came with JRE).

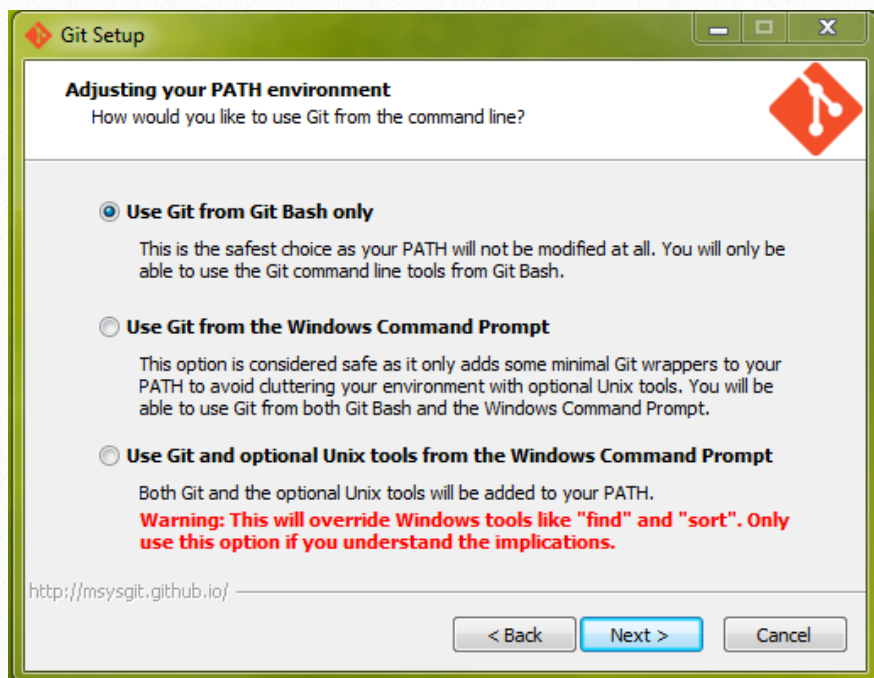
First, run the MsysGit installer .exe. Accept the license agreement, and then you should come to the install location, accept the default should be fine.



which gives you context-sensitive menu when you right click on a file under Git revision control, fine, check that.β



Then you should come to the next page, uncheck the Windows Explorer Integration. If you like Windows Explorer Integration,

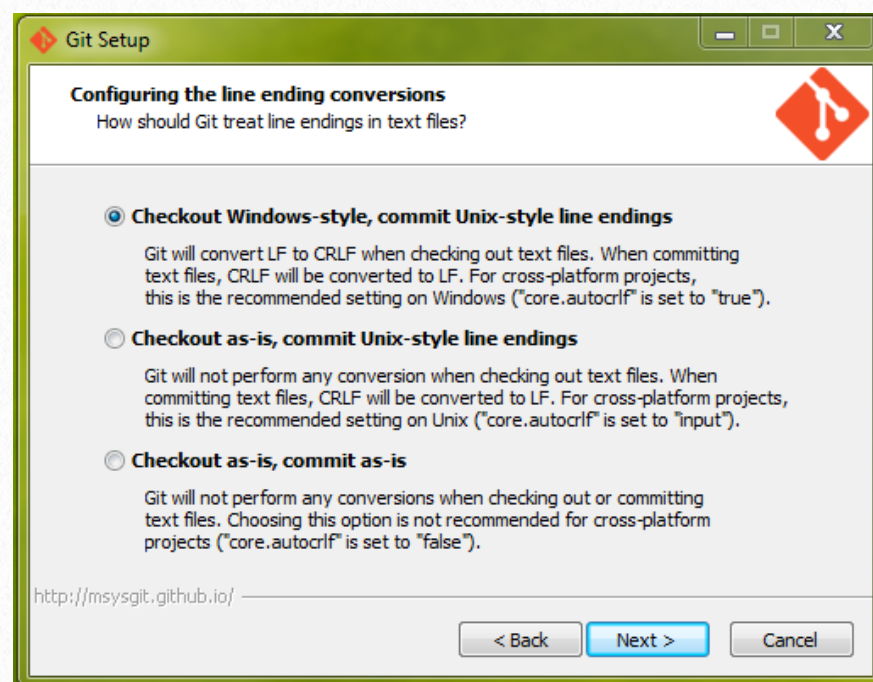


This following screen is important. Make sure you select Checkout Windows-style, commit Unix-style line endings. This is because, the Hempstick Git repository is really hosted on a Unix server, all line endings are stored Unix-style, but Windows applications might not be smart enough to treat Unix-style line ending correctly so all lines are displayed in one row -- impossible to read.

Not only that, if eventually you want to contribute your configuration or code to Hempstick, I will get real pissed if you commit them as Windows-style line ending, filling my screen with the extra distracting CR characters.

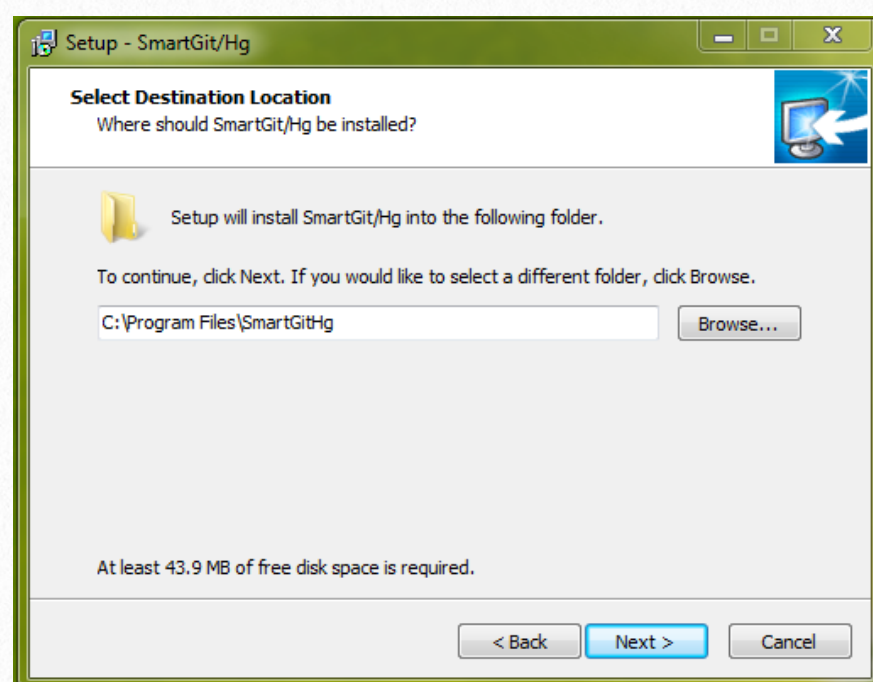
Don't worry, Git will automatically translate them correctly between Windows- and

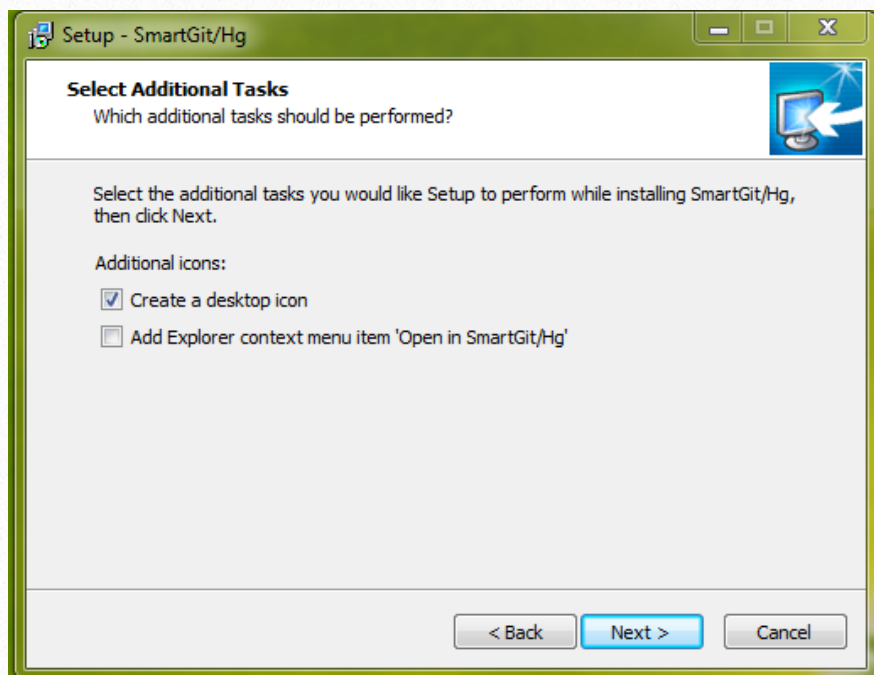
Unix-style line endings during checking in/out.



Now, keep clicking [Next] until you finish the installation.

Next, run the setup.exe for SmartGit.





Keep clicking the [Next] button until it finishes installation. We are not done yet. The first time you run SmartGit, it will ask you a couple of questions.

Go to your desktop and find the newly installed SmartGit icon, and run it. If you already have a GitHub account, in the following SmartGit setup, you may select setting up GitHub as your service provider. Otherwise, select “don’t select a service provider.”

We don’t really care about GitHub’s “fantastic” Cloud-based service at all. I don’t trust my destiny to a “Cloud.” Have you heard it on the news that the Cloud source control hosting company Code Space got hijacked and ransomed and lost control of their Amazon Web Service and perhaps lost a lot the source code hosted there? I

host my own Git server at home, isolated by two layers of both hardware & software firewalls. I only mirror my Git repository up to GitHub for distribution to the public.

Call me old fashioned, or even paranoid if you wish. I don’t know their security policy, backup policy, nor their recovery process, nothing, nada. I don’t trust my source code to people who don’t tell me those! What if they got hacked? Should I audit all my source code and see if backdoors have been planted? Can’t happen? Yeah? SourceForge got hacked twice! Do you want your USB joystick hosting a virus? That would be quite a disaster, wouldn’t it? I mean, that’s the last thing you would suspect. So, you will never find it!

No thank you!

Setup SmartGit/Hg

Steps

- > License Agreement
- Type of Usage
- Git & Hg Executables
- SSH Client
- User Information
- Hosting Providers
- Existing Repositories
- Crash Reporting

License Agreement

Welcome to SmartGit/Hg! Please read the following license agreement carefully. You need to accept it to continue.

License Agreement for SmartGit/Hg

Last date of change: 2014-06-02

1 Subject of the Contract: The license terms of syntevo GmbH (hereinafter called "licensor") are applied for the concession of the rights of use for the entire or partly use of the object code of the software SmartGit/Hg (hereinafter called "SOFTWARE") to contractors, juristic persons under public law or official fund assets in terms of §310 in conjunction with §14 BGB [Civil Code] (hereinafter called "licensee"). Herewith the inclusion of the licensee's own terms and conditions is contradicted, unless their validity has explicitly been agreed to.

☒ I understand and agree to all terms and conditions of this agreement

< Back Next > Finish Exit

Setup SmartGit/Hg

Steps

- License Agreement
- Type of Usage
- Git & Hg Executables
- SSH Client
- > User Information
- Hosting Providers
- Existing Repositories
- Crash Reporting

User Information

User name and e-mail will be stored as part of your commits. Here you can configure the default values which are stored in .gitconfig and .hgrc.

User Name: Your Handle

E-Mail: you@gmail.com

< Back Next > Finish Exit

Setup SmartGit/Hg

Steps

- License Agreement
- > Type of Usage
- Git & Hg Executables
- SSH Client
- User Information
- Hosting Providers
- Existing Repositories
- Crash Reporting

Type of Usage

Choose for what types of repositories you will use SmartGit/Hg.

☐ Free SmartGit/Hg evaluation for commercial use
You may try SmartGit/Hg free of charge for 30 days.

☐ Registered user, commercial use (support) [Purchase](#)
You have purchased a SmartGit/Hg license and are eligible to use SmartGit/Hg for both commercial and non-commercial purposes.
License File:

☒ Non-commercial use only (all features, but no support)
You confirm that you will use SmartGit/Hg solely for non-commercial purposes.

< Back Next > Finish Exit

Setup SmartGit/Hg

Steps

- License Agreement
- Type of Usage
- Git & Hg Executables
- SSH Client
- User Information
- > Hosting Providers
- Existing Repositories
- Crash Reporting

Hosting Providers

If you are using a hosting provider, you can provide account details here to simplify working with your hosted projects.

<Don't configure a hosting provider now>

<Don't configure a hosting provider now>

GitHub

Assembla

Beanstalk


Bitbucket

Codebase

Unfuddle

< Back Next > Finish Exit

Non-Commercial License



Is your use non-commercial?

A use (or purpose) is non-commercial only if it is in no manner primarily intended for or directed toward commercial advantage or private monetary compensation.

When does non-commercial use NOT apply?

Running SmartGit/Hg for 'non-commercial use only' is NOT PERMITTED, if your use is in some sense commercial. Examples of COMMERCIAL uses are:

- you are using SmartGit/Hg to work on your company's projects (no matter whether you are working on open-source projects as well),
- you are a student and you are using SmartGit/Hg for your work as a freelancer,
- you are using SmartGit/Hg in your spare time to manage the website source code of your local football club and you are getting paid for that.

In such cases you will need a commercial license.

Not sure?

Contact sales@syntevo.com and explain your intended use in detail, then we may confirm whether a non-commercial license applies.

☒ I confirm solely non-commercial use

OK Cancel

Setup SmartGit/Hg

Steps

- License Agreement
- Type of Usage
- Git & Hg Executables
- SSH Client
- User Information
- > Hosting Providers
- Existing Repositories
- Crash Reporting

Hosting Providers

If you are using a hosting provider, you can provide account details here to simplify working with your hosted projects.

GitHub

Account:

Token:

[Generate API Token](#)

The (API) token is a special auto-generated credential which SmartGit/Hg will use to authenticate at GitHub. It adds another layer of security, as you can easily revoke access by removing the token from the GitHub front-end.

SmartGit/Hg can create this token for you by connecting to GitHub. If you prefer to not give your password to SmartGit/Hg, you can generate the token yourself from the GitHub front-end (Profile Settings, category 'Applications').

GitHub Enterprise instances can be configured in the Preferences, later.

< Back Next > Finish Exit

Setup SmartGit/Hg

Steps

- License Agreement
- Type of Usage
- Git & Hg Executables
- SSH Client
- User Information
- Hosting Providers
- > Existing Repositories
- Crash Reporting

Existing Repositories

If you already have existing Git or Mercurial repositories, select in what directory they are located, so SmartGit/Hg can find them.

Found repository (selected ones will be remembered):

	Path
<input checked="" type="checkbox"/>	E:\workspace\bf3
<input checked="" type="checkbox"/>	E:\workspace\bf3-o...

Search In: [Start Search](#)

< Back Next > Finish Exit

Setup SmartGit/Hg

Steps

- License Agreement
- Type of Usage
- Git & Hg Executables
- > SSH Client
- User Information
- Hosting Providers
- Existing Repositories
- Crash Reporting

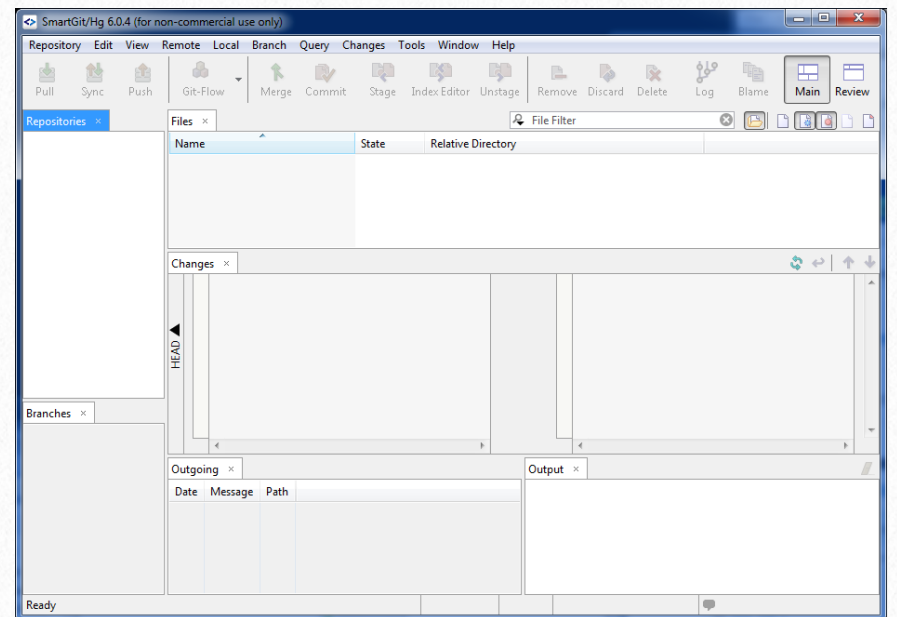
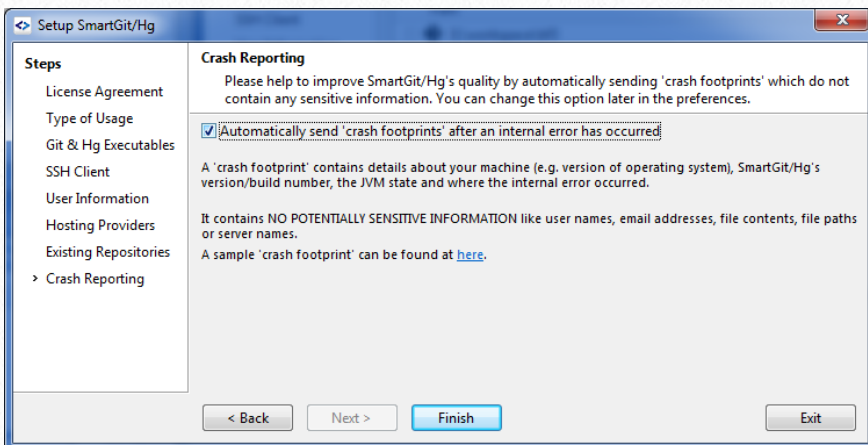
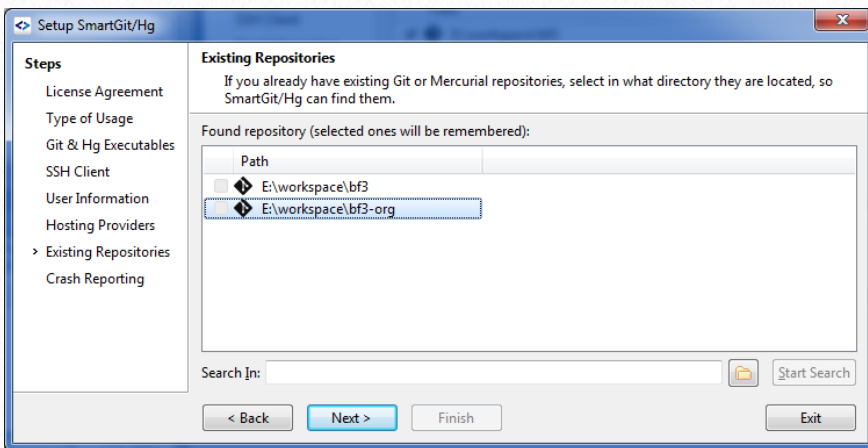
SSH Client

If you are using SSH to connect to other Git repositories, select what SSH client to use. You can change it later in the Preferences.

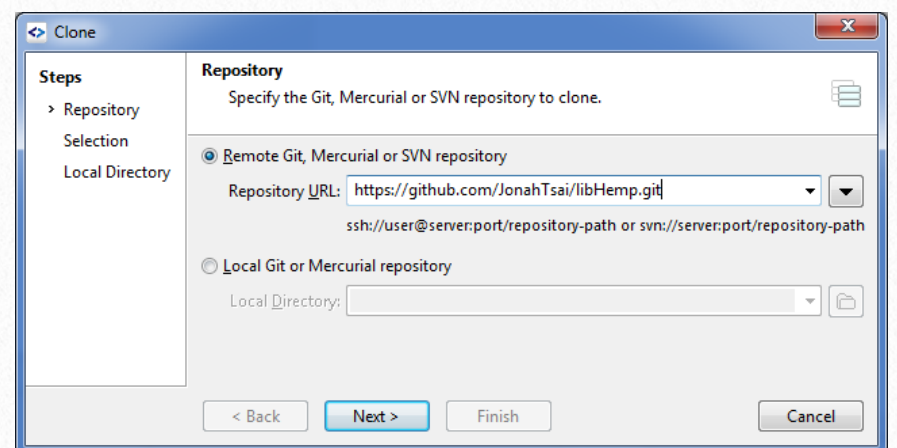
☐ Use system SSH client
may be harder to configure and use for new users, but is more flexible

☒ Use SmartGit/Hg as SSH client
easy to set up and use, supports public key and password authentication

< Back Next > Finish Exit



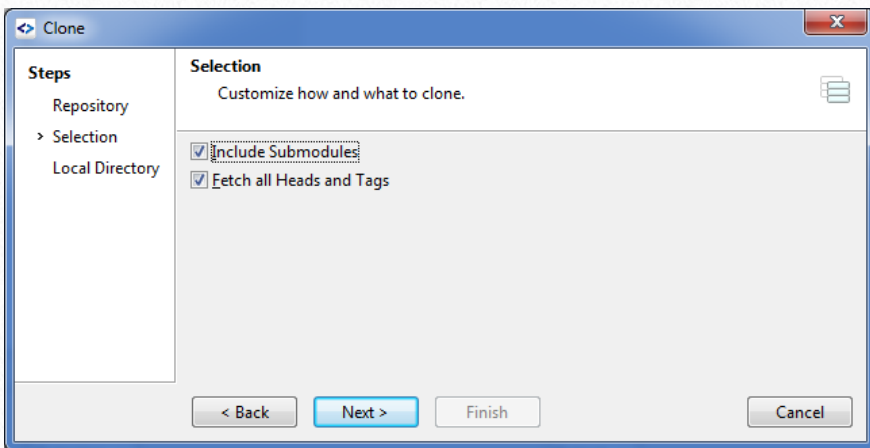
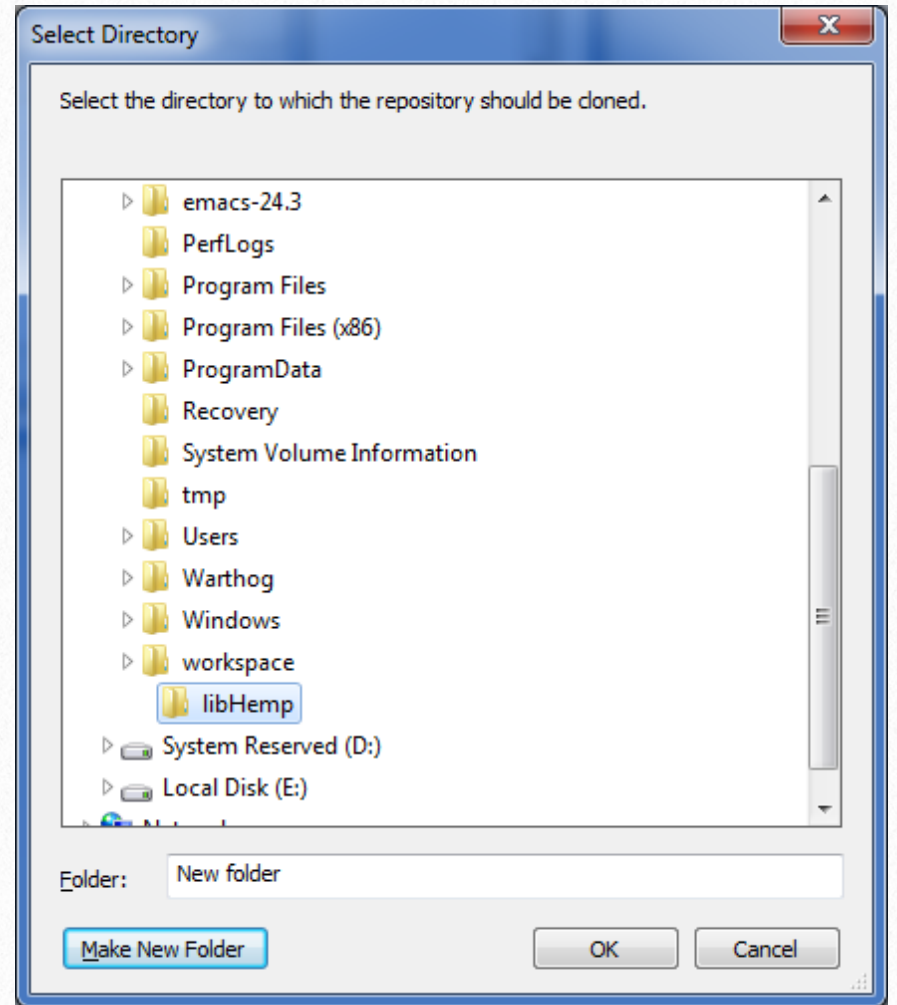
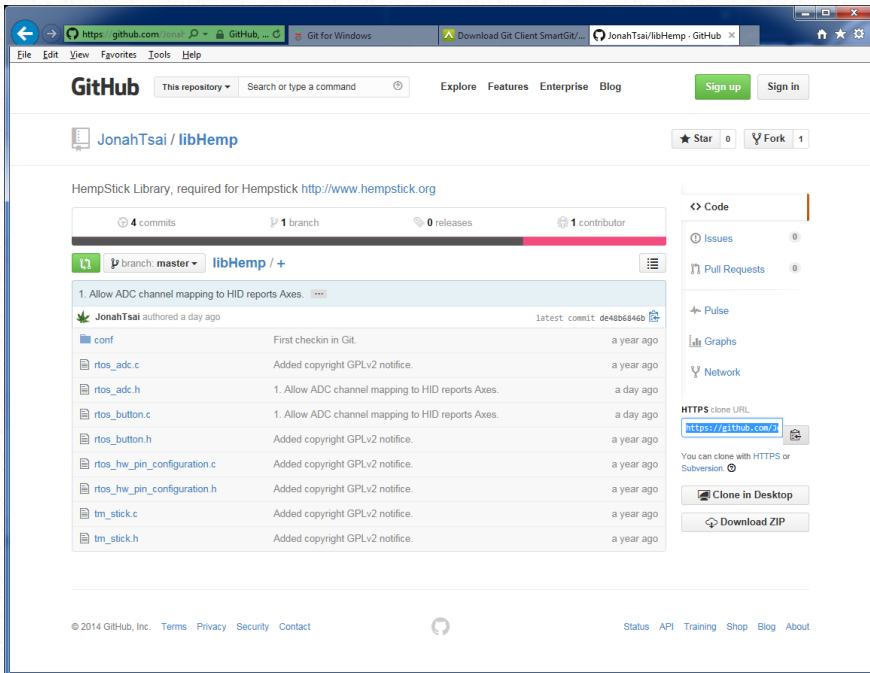
Select the menu [Repository] -> [Clone].



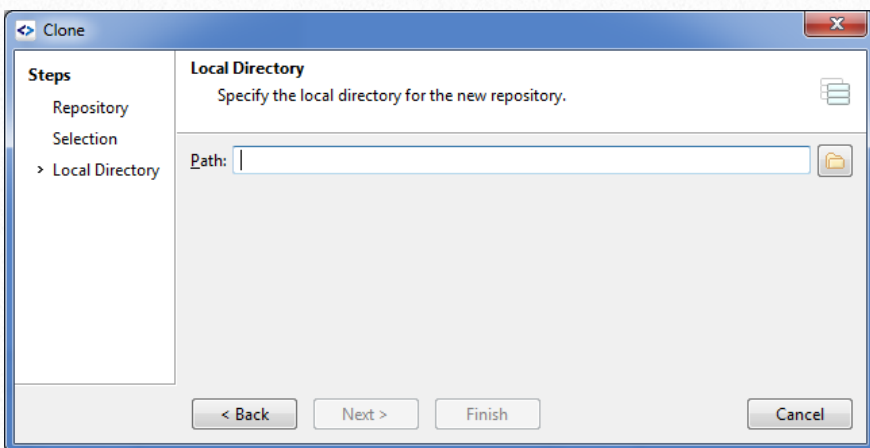
For the last screen, I recommend that you check that “Automatic send ‘crash foot-print....” to help out the author(s) of Smart-Git. It costs you nothing! He’s generous enough to grant us non-commercial uses for free, let’s at least help him back by providing him with crash report so he can improve it!

After clicking on the [Finish] button, we are done with the SmartGit setup. You should see the following screen. Now, on to cloning the libHemp & Hempstick repositories!

We need the URL for libHemp. Launch your favorite browser, and head to <http://www.github.com>. And enter libHemp in the search box. Click on the result link “JonahTsai/libHemp”, and you should come to the next page. On the lower right corner of the screenshot, you should see the highlighted URL. Copy that, and paste into the SmartGit “Repository URL” and click [Next].



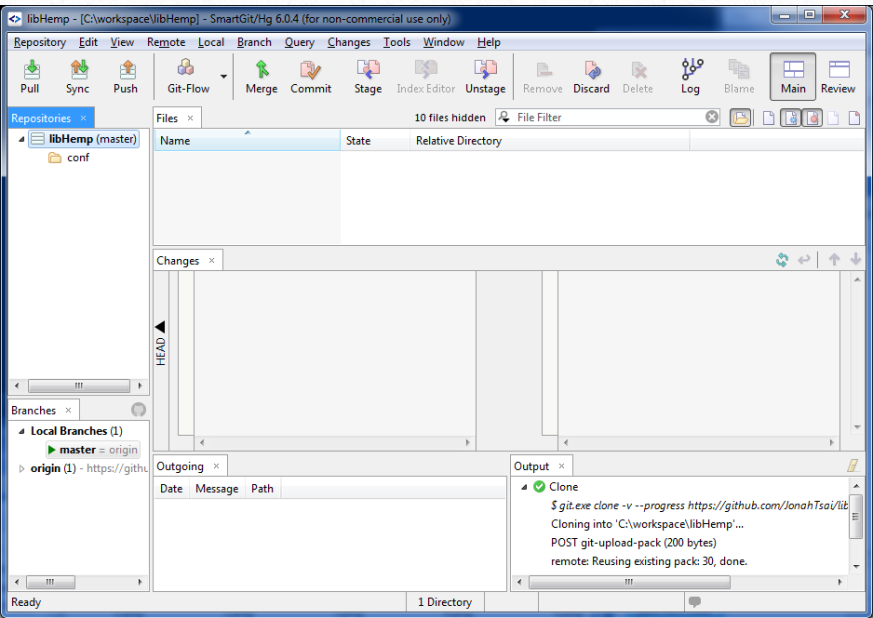
Click the “Browse” button in the next screen.



Select the `c:\workspace\libHemp\` directory. Create the directory if necessary.

You should end up with the following screen. The libHemp repository has been cloned and all the source code and history

are already downloaded and put on your harddrive.



Repeat the same thing to clone the Hempstick repository. And you should end up like the following.

